UNIVERSIDAD CARLOS III MADRID

PROYECTO FIN DE CARRERA
INGENIERÍA INDUSTRIAL

“Desarrollo de una interfaz gráfica de redes neuronales usando Matlab”

AUTOR: ALFONSO MORENO RODRÍGUEZ

TUTOR: ISABEL GONZÁLEZ FARIAS

Índice

Capítulo 1: Introducción y objetivos.

1.1.-Introducción.

Las redes neuronales son técnicas no paramétricas muy utilizadas en diversos ámbitos de la ciencia e ingeniería porque permiten resolver problemas complejos, que muchas veces no son fáciles de resolver utilizando técnicas tradicionales como la regresión lineal o polinómica. Las redes neuronales permiten obtener un modelo no explícito que relaciona un conjunto de variables salida con un conjunto de variables entrada. Así, estos modelos permiten predecir cuál es el valor de salida, dados unos valores de entrada del modelo. Para estimar el modelo es necesario disponer de un conjunto de observaciones de las variables. Estas observaciones son usadas como patrones de entrenamiento para que la red aprenda y sea capaz de predecir una salida del modelo, ante nuevas observaciones. Por tanto, las capacidades de la red van a depender en gran medida de esta fase de entrenamiento. En la fase de entrenamiento es necesario controlar muchos parámetros y distintos algoritmos de optimización, por lo que el usuario de una red neuronal debe tener conocimiento suficiente de cuáles son estos parámetros y cómo funcionan. Por otro lado, una vez entrenada la red, es muy importante también evaluar la robustez del modelo creado, comprobando que es adecuado para nuevos datos. Es importante, realizar un buen análisis de los resultados obtenidos.

Existen muchos tipos diferentes de redes neuronales, pero en este proyecto sólo se trabajarán con dos de ellas, las redes backpropagation (o perceptron multicapa) y las redes de base radial, por ser éstas las más utilizadas en la práctica.

1.2.-Objetivos.

En este proyecto se desarrollará una aplicación dirigida, principalmente, a usuarios que no han trabajado nunca con redes neuronales o no muy expertos, con el fin de que puedan realizar el entrenamiento y evaluación de redes neuronales backpropagation y de base radial de manera sencilla y cómoda. Dicha aplicación se desarrollará utilizando herramientas de Matlab, tanto las de la toolbox de redes neuronales, como las herramientas para la programación de ventanas gráficas que dispone (GUI).

Esta aplicación además, pretende ser un herramienta docente para la asignatura "Técnicas Avanzadas de Diseño en Ingeniería Mecánica", impartida en el máster de Ingeniería de Máquinas y Transportes de la Universidad Carlos III de Madrid. En esta asignatura, se imparte, en un total de 4 horas, los conceptos teóricos de los dos tipos de red mencionados y se incluye una parte práctica. Sin embargo, actualmente debido al poco tiempo del que se dispone resulta un poco difícil transmitir al alumno el uso práctico de estos modelos. Esta

herramienta, facilitará en gran medida que los alumnos comprendan cuáles son las etapas en el desarrollo de una red neuronal y cuáles son los aspectos más importantes que deben tener en cuenta, como por ejemplo, cómo tratar los datos, cómo analizar los resultados, etc.

Por ello, en el desarrollo de la aplicación se ha perseguido que sea fácil de utilizar, que no sea necesario tener vastos conocimientos de redes neuronales y que la visualización de resultados sea clara y concisa. Además se ha desarrollado una ayuda donde se explica los fundamentos de los principios de redes neuronales utilizados en la aplicación y cómo se utiliza el programa. Todo ello permite al usuario disponer de una herramienta sencilla y clara para trabajar con redes neuronales por primera vez y conseguir la experiencia necesaria para ser capaz de trabajar, más adelante, con programas más complejos.

1.3.-Estructura del proyecto

El proyecto ha sido dividido en 7 capítulos. El capítulo II incluye una breve descripción de los conceptos teóricos más importantes de redes neuronales. En los capítulo 3 y 4 se describen las herramientas de la toolbox de redes neuronales de Matlab que se van a utilizar, y las herramientas de Matlab para crear una GUI. Estos capítulos tienen como objetivo servir de soporte al Capítulo 5, capítulo principal del proyecto, en el que se describe detalladamente el modo de utilización y justificación de la aplicación desarrollada. En el capítulo 6, se incluyen dos casos a modo de ejemplo para que el lector entienda las posibilidades de la aplicación y su modo de utilización. Por último, en el Capítulo 7 se concluye acerca de lo realizado en el proyecto y se definen las posibles líneas de mejora para la aplicación teniendo claro su principal objetivo, que no es otro que el de servir como herramienta de redes neuronales para usuarios no iniciados en la materia.

CAPÍTULO 2: Fundamentos teóricos de las redes neuronales

2.1 Fundamentos biológicos de las redes neuronales.

Las redes neuronales artificiales se basan en el funcionamiento del sistema neuronal del cuerpo humano. En el cuerpo humano encontramos 3 elementos fundamentales: los órganos receptores que recogen información del exterior; el sistema nervioso que transmite la información, la analiza y en parte almacena, y envía la información elaborada y, los órganos efectores que reciben la información de parte del sistema nervioso y la convierte en una cierta acción.

La unidad fundamental del sistema nervioso es la neurona. Las neuronas se unen unas con otras formando redes. Se componen de un cuerpo o núcleo, del axón, que es una ramificación de salida de la neurona, y de un gran número de ramificaciones de entrada llamadas dendritas. Su funcionamiento es el siguiente. Las señales de entrada llegan a la neurona a través de la sinapsis, que es la zona de contacto entre neuronas (u otro tipo de células, como las receptoras). La sinapsis recoge información electro-química procedente de las células adyacentes que están conectadas a la neurona en cuestión. Esta información llega al núcleo de la neurona, a través de las dendritas, que la procesa hasta generar una respuesta, la cual es posteriormente propagada por el axón.

La sinapsis está compuesta de un espacio líquido donde existe una cierta concentración de iones. Este espacio tiene unas determinadas características eléctricas que permiten inhibir o potenciar la señal eléctrica a conveniencia.

Por ello, se puede ver que el sistema neuronal es un conjunto de neuronas conectadas entre sí, que reciben, elaboran y transmiten información a otras neuronas, y que dicha información se ve potenciada o inhibida en la siguiente neurona a conveniencia, gracias a las propiedades del espacio intersináptico.

De hecho, esta propiedad de poder alterar el peso de cada información en la red neuronal nos otorga en cierta medida la capacidad de aprender.

2.2 Redes neuronales artificiales

Las redes neuronales artificiales tratan de emular las características y propiedades de las redes neuronales biológicas. En general, consisten en una serie de unidades denominadas neuronas, conectadas entre sí.

Cada neurona recibe un valor de entrada, el cual transforma según una función específica denominada *función de activación*. Dicha señal transformada pasa a ser la salida de la neurona.

Las neuronas se conectan entre sí según una determinada arquitectura. Cada conexión tiene un determinado *peso* que pondera cada entrada a la neurona. De esta manera la entrada de cada neurona es la suma de las salidas de las neuronas conectadas a ella, multiplicadas por el peso de la respectiva conexión. La figura siguiente ilustra dicho concepto:

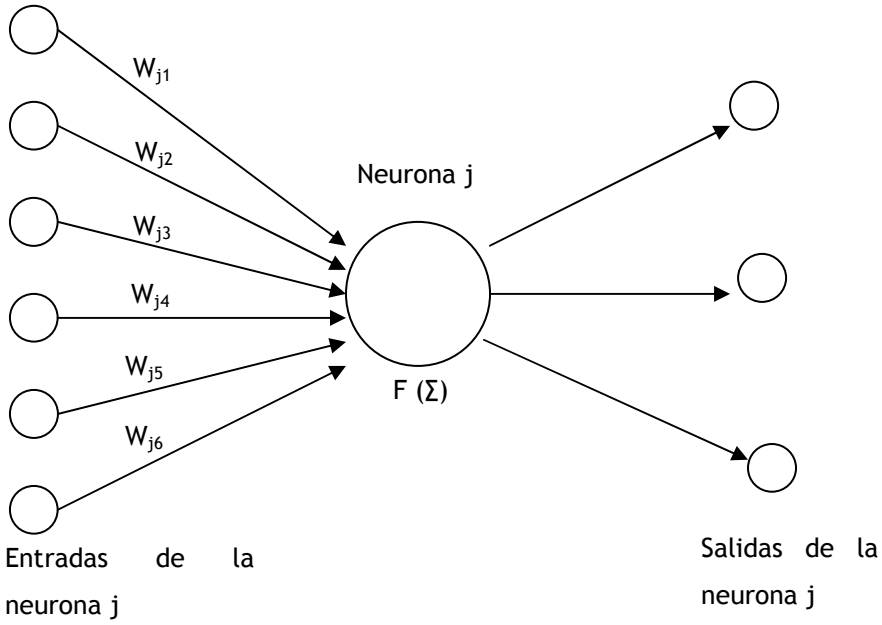

Figura 2.1 Esquema de funcionamiento de una neurona

En este modelo, la neurona j recibe una serie de entradas x_1, x_2, …, x_n. Cada señal se multiplica por el peso asociado a su conexión, w_1, w_2,…,w_n. Luego, se suman estas entradas ponderadas y se les aplica la función de activación F(.) para generar la señal de salida de la neurona j. Los valores de los pesos son ajustados durante la fase de aprendizaje.

Como se ha comentado anteriormente, estas neuronas están conectadas entre sí de acuerdo a una determinada *arquitectura*. Es decir, las neuronas se agrupan en distintas capas: *una capa de entrada, otra de salida*, y en el caso de existir, una o varias *capas ocultas*. La salida de cada neurona se propaga por igual por estas conexiones hasta las neuronas de destino. Cada conexión tiene un peso asociado que pondera el valor numérico de la señal que viaja por ésta. Así pues, una red de neuronas artificial puede verse como un grafo cuyos nodos tienen funcionamiento similar, los cuales propagan la información a través de las distintas conexiones.

Veamos el funcionamiento de una red. Para ello nos referimos a la figura 2.2. Las entradas a la red son introducidas en las neuronas de la capa de entrada, que normalmente genera una salida tal cual o las escala para que las señales se encuentren en un determinado rango. Estas entradas son propagadas a las neuronas de la siguiente capa. De acuerdo al esquema de la figura 2.1 cada neurona j de la segunda capa generará una salida de valor:

$$S_{2j} = F_{2j}(X_1 W_{1j}) \qquad (2.1)$$

Donde X_1 es el vector de entradas de la capa 1 y W_{1j} el vector de pesos correspondientes a las conexiones que van de todas las neuronas de la primera capa a la neurona j de la segunda capa. La función F_{2j} es la función de activación de la neurona j de la segunda capa. Así con todas las neuronas de la segunda capa. Estas salidas son propagadas a las neuronas de la capa de salida. Estas neuronas generan las salidas de la red. Cada neurona i de la capa de salida generará una salida de valor:

$$S_{si} = F_{si}(W_{2i} S_2) \qquad (2.2)$$

Donde W_{2i} es el vector de pesos correspondientes a las conexiones que van de las neuronas de la segunda capa a la neurona i de la capa de salida, y S_2 el vector de salidas de las neuronas de la capa dos, que a su vez son entradas de las neuronas de la capa de salida.

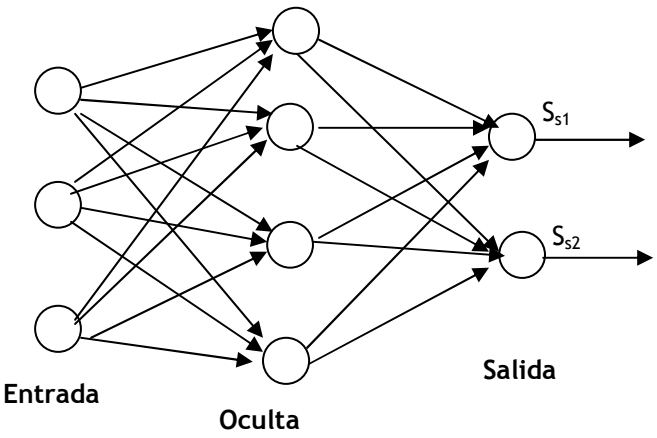

Figura 2.2: Esquema de la arquitectura de una red

Por último hablar del aspecto más importante y delicado de redes neuronales, el aprendizaje. Las RNA son sistemas de aprendizaje basadas en datos que son utilizados como patrones. Por ello la capacidad de una red de resolver un problema está muy ligada a los patrones utilizados durante su fase de aprendizaje.

El *aprendizaje* de una red neuronal consiste en hallar los valores precisos de los pesos de sus conexiones para que pueda resolver un determinado problema. El proceso general consiste en ir introduciendo una serie de datos patrón y ajustar los pesos siguiendo un determinado criterio. Los criterios que se van a utilizar en este proyecto se fundamentan en el error cometido por la red, lo que nos obliga a conocer la salida que se debería obtener para cada uno de ellos. Es lo que se conoce como entrenamiento supervisado. De esta manera, primero se introducen los patrones, se reajustan los pesos, posteriormente se comprueba si se ha cumplido un determinado criterio de convergencia, de no ser así se repite todo el proceso.

2.3 Redes Perceptron Multicapa

2.3.1 Introducción

El perceptron multicapa con conexiones hacia adelante es una generalización del perceptron simple. Surge como respuesta a los problemas que tenía dicha red, como por ejemplo, no poder resolver problemas que no fueran linealmente separables. De hecho, algunos autores han demostrado que el perceptron multicapa es un aproximador universal de cualquier función continua en el espacio $\mathbb{R}^n$.

2.3.2 Arquitectura del perceptron multicapa

La arquitectura de este tipo de red se caracteriza porque tiene todas sus neuronas agrupadas en distintos niveles llamados capas. El primer nivel corresponde a la capa de entrada, que se encarga únicamente de propagar por el resto de la red las entradas recibidas. El último nivel es el de la capa de salida. Se encarga de proporcionar los valores de salida de la red. En las capas intermedias denominadas capas ocultas, se realiza un procesamiento no lineal de los patrones recibidos.

Las conexiones del perceptron multicapa son hacia adelante. Generalmente todas las neuronas de un nivel se conectan con todas las neuronas de la capa inmediatamente posterior. A veces, dependiendo de la red, se encuentran conexiones de neuronas que no están en niveles consecutivos, o alguna de las conexiones entre dos neuronas de niveles consecutivos no existe, es decir, el peso asociado a dicha conexión es constante e igual a cero. Además, todas las neuronas de la red tienen un valor umbral asociado. Se suele tratar como una entrada cuyo valor es constante e igual a uno, y lo único que varía es el peso asociado a dicha conexión (que es el umbral realmente).

Por otro lado, las funciones de activación que se suelen utilizar son la función identidad, la función sigmoidal y la función tangente hiperbólica. A continuación se muestran sus respectivas expresiones.

Función identidad

$$f_1(x) = x \qquad (2.3)$$

Función sigmoidal

$$f_2(x) = \frac{1}{1 + e^{-x}} \qquad (2.4)$$

Función tangente hiperbólica

$$f_3(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \qquad (2.5)$$

La principal diferencia entre la función sigmoidal y la función tangente hiperbólica es el rango de sus valores de salida. Mientras que para la primera su rango es [0,1], para la segunda es [-1,1]. De hecho existe una relación entre las dos. Se relacionan mediante la expresión $f_2(x) = 2f_1(x) - 1$, por lo que la elección entre una u otra se elige en función del recorrido que interese.

2.3.3. Algoritmo de retropropagación

Este algoritmo es el algoritmo básico de aprendizaje que usa el perceptron multicapa. Es el algoritmo mediante el cual se van adaptando todos los parámetros de la red. El tipo de entrenamiento que sigue este tipo de red es supervisado.

El aprendizaje de la red se plantea como un problema de minimización de una determinada función de error. En general se usa como función del error, el error medio cuadrático, es decir:

$$E = \frac{1}{N}\sum_{n=1}^{N} e(n) \qquad (2.6)$$

$$e(n) = \frac{1}{2} \cdot \sum_{n=1}^{t} (s(n) - y(n))^2 \qquad (2.7)$$

Donde *s(n)* es la salida patrón, *y(n)* la salida obtenida de la red, t el número de neuronas de salida y N el número de patrones.

El problema es no lineal y como tal, el problema de minimización de la función error se resuelve por técnicas de optimización no lineales que se basan en ajustar los parámetros siguiendo una determinada dirección. En este método, la dirección elegida es la negativa del gradiente de la función error. Más adelante veremos que existen otros métodos que no se basan en esta dirección de cambio.

A partir de aquí existen dos opciones. Podemos cambiar los parámetros cada vez que introducimos el patrón, o solamente cambiarlos cuando hayamos introducido todos los parámetros de entrenamiento por cada ciclo. En el primer caso, debemos minimizar *e(n)* y en el segundo se minimiza la función E. A continuación se presentará el desarrollo para el primer caso, pero la extensión al segundo es inmediata.

De acuerdo a lo que hemos dicho antes, todos los pesos deben variar según la dirección del gradiente del error. Matemáticamente esto se expresa de la siguiente forma:

$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w} \qquad (2.8)$$

Por lo tanto el problema consiste en evaluar la derivada. El parámetro α es la tasa de aprendizaje que influye en la magnitud del desplazamiento en la superficie de la función error. A continuación se evalúa el valor del gradiente para los pesos de las conexiones de la última capa oculta a las neuronas de la capa de salida.

Sea $w_{ji}^{c-1}(n)$ el peso de la conexión de la neurona j de la capa C-1 a la neurona i de la capa de salida. El error $e(n)$ viene dado por la ecuación (2.7), y además éste solo se ve afectado por w_{ji}^{c-1} en el error de la salida de la neurona i. Por tanto:

$$\frac{\partial e(n)}{\partial w_{ji}^{c-1}} = -(s_i(n) - y_i(n)) \frac{\partial y_i(n)}{\partial w_{ji}^{c-1}} \qquad (2.9)$$

Por otro lado, la salida de la neurona i es igual a la suma de las entradas transformadas según su función de activación. Aplicando la regla de la cadena, y teniendo en

cuenta que w_{ji}^{c-1} solo afecta a la entrada de la neurona i porque va multiplicando a_j se tiene que:

$$\frac{\partial y_i(n)}{\partial w_{ji}^{c-1}} = f'\left(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} a_j^{c-1} + u_i^c\right) a_j^{c-1}(n) \qquad (2.10)$$

Se define δ asociado a la neurona i de la capa C del patrón n, $\delta_i^c(n)$, del siguiente modo:

$$\delta_i^c(n) = -\left(s_i(n) - y_i(n)\right) f'\left(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} a_j^{c-1} + u_i^c\right) \qquad (2.11)$$

De tal manera que el gradiente se exprese ahora mediante la siguiente expresión:

$$\frac{\partial e(n)}{\partial w_{ji}^{c-1}} = \delta_i^c(n) a_j^{c-1}(n) \qquad (2.12)$$

Esto es extensible para todos los pesos de las conexiones de las neuronas de la capa C-1 con las neuronas de la capa de salida. Se procede de la misma forma para los valores umbrales obteniéndose una expresión análoga.

A continuación se procede a calcular la magnitud de la variación para el resto de conexiones que se existe entre las capas ocultas. Se va a proceder de la misma forma que hasta ahora y se obtendrá una ley de recurrencia para la modificación de dichos pesos. Consideremos el peso w_{kj}^{c-2} de la conexión de la neurona k de la capa C-2 con la neurona j de la capa C-1. El nuevo valor de dicho peso vendrá dado por una expresión análoga a la dada en (2.8), donde lo único que se desconoce es el valor del gradiente. En este caso w_{kj}^{c-2} influye en todas las salidas de la red, por lo que se tiene que:

$$\frac{\partial e(n)}{\partial w_{kj}^{c-2}} = -\sum_{i=1}^{n_c} \left(s_i(n) - y_i(n)\right) \frac{\partial y_i(n)}{\partial w_{kj}^{c-2}} \qquad (2.13)$$

Para calcular la derivada hay que tener en cuenta que la salida w_{kj}^{c-2} influye en la entrada de la neurona j (por lo que influye en su salida), que a su vez influye en la entrada de todas las neuronas de salida. Por tanto, aplicando la regla de la cadena tenemos:

$$\frac{\partial y_i(n)}{\partial w_{kj}^{c-2}} = f'\left(\sum_{j=1}^{n_{c-1}} w_{ij}^{c-1} a_j^{c-1} + u_j^{c-1}\right) \frac{\partial a_j^{c-1}}{\partial w_{kj}^{c-2}} \qquad (2.14)$$

Por otro lado la salida de la neurona j, únicamente se ve afectada por w_{kj}^{c-2} que va multiplicando a a_k^{c-2}. Por lo tanto, y volviendo a aplicar la regla de la cadena tenemos:

$$\frac{\partial a_j^{c-1}}{\partial w_{kj}^{c-2}} = f'\left(\sum_{k=1}^{n_{c-2}} w_{kj}^{c-2} a_k^{c-2} + u_k^{c-2}\right) a_k^{c-2}(n) \tag{2.15}$$

Definiendo el valor de δ para las neuronas de la capa C-1 de la siguiente forma:

$$\delta_j^{c-1}(n) = f'\left(\sum_{k}^{n_{c-2}} w_{kj}^{c-2} a_k^{c-2} + u_j^{c-2}\right) \sum_{i=1}^{n_c} \delta_i^c(n) w_{ji}^{c-1} \tag{2.16}$$

Se tiene que el valor de la derivada del error con respecto al peso w_{kj}^{c-2} es:

$$\frac{\partial e(n)}{\partial w_{kj}^{c-2}} = \delta_j^{c-1}(n) a_k^{c-2}(n) \tag{2.17}$$

Por lo que el nuevo valor de dicho peso viene dado por la expresión:

$$w_{kj}^{c-2}(n) = w_{kj}^{c-2}(n-1) - \alpha \delta_j^{c-1}(n) a_k^{c-2}(n) \tag{2.18}$$

Se observa que la modificación de los pesos de las conexiones entre las neuronas de la capa $C-2$ y las neuronas de la capa $C-1$ se ve afectada por la salida de la neurona k de la capa $C-2$ y el término δ asociado a la neurona a la que llega la conexión. Lo único que varía es la expresión de δ. En este punto es posible generalizar la actualización de cualquiera de los pesos de cualquier capa dentro de la red, acorde a la ecuación (2.18). Por ello, la actualización de uno de los pesos de la conexión que une la neurona k de la capa h con la neurona j de la capa $h+1$ vendrá dado por la expresión:

$$w_{kj}^{h}(n) = w_{kj}^{h}(n-1) - \alpha \delta_j^{h+1}(n) a_k^{h}(n) \tag{2.19}$$

Donde el término δ viene dado por la siguiente ley de recurrencia:

$$\delta_j^{h+1} = f'\left(\sum_k^{n_h} w_{kj}^h a_k^h + u_j^{h+1}\right) \sum_{i=1}^{n_{h+2}} \delta_i^{h+2}(n) w_{ji}^{h+1} \qquad (2.20)$$

El término δ propaga los errores obtenidos a la salida hacia atrás. De esta manera, cada neurona oculta recibe un cierto error o valor δ de todas las neuronas a las cuales se conecta, y la suma de todos estos errores es una medida del error total que comete esta neurona. Para la actualización de los valores umbrales se procede de la misma forma llegando a una expresión para su actualización dado por la siguiente expresión:

$$u_j^{h+1}(n) = u_j^{h+1}(n-1) - \alpha\delta_j^{h+1}(n) \qquad (2.21)$$

En este tipo de entrenamiento se define el número de épocas como el número de veces que se han comparado los ejemplos con las salidas de la red para realizar los ajustes en los pesos de las conexiones.

2.3.4 Mejoras del algoritmo de aprendizaje

En el apartado anterior se ha descrito el algoritmo de retropropagación, que es el algoritmo más conocido para el entrenamiento de redes neuronales, pero que en la práctica su uso se ve muy limitado debido a que su convergencia es muy lenta. Por ello se han introducido mejoras de este algoritmo para acelerar la convergencia del mismo. En esta sección describiremos someramente algunos de ellos.

- ***Regla delta generalizada con momentos***

La regla delta generalizada se basa en la búsqueda del mínimo de la función error mediante el descenso por la superficie a través del gradiente de la misma. Esto nos puede llevar a un mínimo local de la función error, donde el gradiente vale cero, y por lo tanto los pesos no se ven modificados, pero el error cometido por la red es significativo. Para evitar este fenómeno este método propone que los cambios de los pesos de las conexiones tengan en cuenta el gradiente de la iteración anterior, realizando un promedio de éste y el actual. De esta manera, se desciende por una dirección intermedia entre el gradiente actual y el de la iteración anterior. Matemáticamente esto queda expresado por la expresión:

$$w(n) = w(n-1) - \alpha\frac{\partial e(n)}{\partial w} + \eta\Delta w(n-1) \qquad (2.22)$$

El término $\Delta w(n-1)$ es la magnitud del cambio de la iteración anterior. Es donde se tiene en cuenta el gradiente anterior. El parámetro η es el momento del algoritmo. El efecto de la inclusión del momento en este algoritmo es tal que acelera el cambio si las direcciones de las dos iteraciones son similares, y realiza pasos más pequeños si se producen oscilaciones en el gradiente, es decir tiene un efecto estabilizador.

- ***Tasa de aprendizaje variable***

La tasa de aprendizaje α juega un papel muy importante en el comportamiento de estos algoritmos de aprendizaje. Si es pequeño, la magnitud del cambio de los pesos sinápticos será pequeña y por lo tanto tardará mucho en converger. Si es demasiado grande el algoritmo oscilará y no encontrará nunca un mínimo de la función error. Se demuestra que el valor óptimo de la tasa de aprendizaje para una convergencia rápida es el valor inverso del mayor autovalor de la matriz hessiana H. Pero computacionalmente este proceso es ineficiente, ya que para obtener la matriz H es necesario evaluar las derivadas segundas de la función error. Por ello se emplea técnicas heurísticas que van variando el valor de la tasa de aprendizaje en cada iteración. Algunas de éstas son:

- Incrementar la tasa de aprendizaje cuando $\nabla E(n-1)$ es próximo a $\nabla E(n)$ y disminuir la en caso contrario.
- Multiplicar la tasa de aprendizaje por $a > 1$ si $\frac{\partial E(n-1)}{\partial w}$ y $\frac{\partial E(n)}{\partial w}$ tienen el mismo signo, o por $b \in (0,1)$ en caso contrario.
- Multiplicar la tasa de aprendizaje por una cantidad mayor que uno cuando haya decrecido la función error con el fin de avanzar más rápidamente, y multiplicarla por una cantidad menor que uno en caso contrario. Este heurístico es el que utiliza Matlab en el entrenamiento *traingda* (ver apartado 31.3).

- ***Método elástico de retropropagación.***

Cuando las entradas a las neuronas de una red son muy grandes, y la función de activación de dichas neuronas es una función sigmoidea, el algoritmo de retropropagación de errores o regla delta generalizada sufre un bloqueo, es decir los pesos sináptico de las conexiones apenas varían debido a la magnitud del gradiente. Se produce dicho fenómeno porque la derivada de la función sigmoidea es cercana a cero cuando el argumento de dicha función es grande. Para evitarlo se utiliza este método que no determina la magnitud de la variación con el valor del gradiente, aunque si utiliza su signo para ver la dirección de cambio. El tamaño del cambio del peso está determinado exclusivamente por un valor específico conocido como valor de actualización Δ_{ij}. Matemáticamente esto se expresa por:

$$w(n) = \begin{cases} w(n-1) - \Delta_{ij} & \text{si} \quad \frac{\partial E(n)}{\partial w} > 0 \\ w(n-1) + \Delta_{ij} & \text{si} \quad \frac{\partial E(n)}{\partial w} < 0 \\ w(n-1) & \text{si} \quad \frac{\partial E(n)}{\partial w} = 0 \end{cases} \tag{2.23}$$

-Métodos de mayor orden de convergencia

La rapidez de estos métodos es del orden de 10 a cientos de veces más rápidos que los anteriores. Se basan en la idea de que la dirección del gradiente no tiene porque ser la que más rápido converja hacia el mínimo de la función error. De esta manera, estos métodos actualizan los pesos de acuerdo con esta expresión:

$$w(n) = w(n-1) + \alpha(n)d(n) \tag{2.24}$$

Donde $d(n)$ indica la dirección de búsqueda de la iteración n. Esta dirección es la conjugada de las $n-1$ direcciones anteriores. Se dice que un conjunto de vectores no nulos son conjugados con respecto a una matriz A si son linealmente independientes y $d(i)^t Ad(j) = 0, \ \forall i \neq j$.

Si utilizamos el desarrollo de Taylor generalizado cuadrático para aproximar la función E en el entorno del punto $\overrightarrow{w_o}$ y derivamos para obtener su mínimo obtenemos:

$$E(\vec{w}) = E(\overrightarrow{w_0}) + \nabla E(\overrightarrow{w_0})^T(\vec{w} - \overrightarrow{w_0}) + \frac{1}{2}(\vec{w} - \overrightarrow{w_0})^T H(w_0)(\vec{w} - \overrightarrow{w_0}) \tag{2.25}$$

$$\nabla E(w) = \nabla E(w_0) + H(w_0)(w - w_0) = 0 \tag{2.26}$$

Donde H es la matriz Hessiana. Si existe la inversa de H entonces se puede hallar una fórmula recurrente para la actualización de los pesos despejando de la ecuación 2.26. Es decir:

$$w(n) = w(n-1) - H\big(w(n-1)\big)^{-1}\nabla E(w(n-1)) \tag{2.27}$$

En general, el método de Newton puede que no converja, si el punto inicial no está lo suficientemente cerca del óptimo. Además requiere el cálculo de la inversa de la matriz Hessiana que requiere un gran esfuerzo computacional, por lo que en la práctica este método

no se utiliza. El método de la secante evalúa la matriz Hessiana por medio del cálculo de la secante de la curva para evitar el mayor esfuerzo computacional que requiere el cálculo de las derivadas segundas.

Existen modificaciones del método de Newton para que pueda converger partiendo de cualquier punto. Uno de ellos (algoritmo de Fletcher-Reeves) es generar direcciones conjugadas mediante la regla:

$$\boldsymbol{d_n = -\nabla E(w_n) + \beta(n) d_{n-1}} \tag{2.28}$$

Donde $\beta(n)$ viene dado por la expresión:

$$\boldsymbol{\beta(n) = \frac{g_n^T g_n}{g_{n-1}^T g_{n-1}}} \tag{2.29}$$

Donde $\boldsymbol{g}$ son los gradientes de la función error de las respectivas iteraciones. Una vez obtenido la dirección de variación de los pesos sinápticos por la ecuación (2.28) con ayuda de la expresión dada en (2.29) los nuevos pesos vendrán dados por:

$$\boldsymbol{w(n) = w(n-1) + \alpha d(n)} \tag{2.30}$$

-Algoritmo de Levenberg-Marquardt

Es una modificación del método de Newton que elimina el cálculo de la matriz Hessiana. Requiere mayor esfuerzo computacional que los métodos de descenso por el gradiente pero es mucho más rápido que éstos. De hecho es el algoritmo que Matlab utiliza por defecto para entrenar las redes neuronales.

Como ya se comento, el problema de actualización de los pesos se resuelve minimizando la función error (se utiliza el error medio cuadrático), es decir:

$$\min\big(E(w)\big) = \min\left(\frac{1}{2}\sum_{i=1}^{N}(s_i - y(w)_i)^2\right) \tag{2.31}$$

La función error se puede expresar de la siguiente forma:

$$E(w) = \frac{1}{2} e(w)^T e(w) \tag{2.32}$$

donde:

$$e(w)^T = (e_1(w), e_2(w), \ldots\ldots, e_N(w)) \tag{2.33}$$

$$e_i(w) = s_i - y(w)_i \tag{2.34}$$

La derivada de la función error $\nabla E(w)$ tiene la siguiente forma:

$$\nabla E(w) = e(w)^T \nabla e(w) = J(w)^T e(w) \tag{2.35}$$

Asimismo, la matriz Hessiana vendrá dada por:

$$H(w) = \sum_{i=1}^{N} \left(\nabla e_i(w) (\nabla e_i(w))^t + e_i(w) \nabla^2 e_i(w) \right) = J(w)^t J(w) + S(w) \tag{2.36}$$

Por otro lado, si tenemos en cuenta que la magnitud del cambio de los pesos sinápticos viene dada por la expresión:

$$\Delta w = -\alpha M(w) \nabla E(w) \tag{2.37}$$

Donde si $M(w) = I$ tenemos el método del descenso en la dirección del gradiente y si $M(w) = H^{-1}(w)$ nos encontramos con el método de Newton. En general, será más conveniente llegar a un compromiso entre estos dos métodos. Para ello se toma $M(w) = [\mu I + H(w)]^{-1}$. De esta manera la dirección del cambio de los pesos sinápticos vendrá dado por:

$$d(n) = -\left[\mu I + H(w(n))\right]^{-1} \nabla E(w(n)) \tag{2.38}$$

Debido a la dificultad de evaluar la matriz Hessiana, evaluamos solamente su término de primer orden $J(w)^t J(w)$, suponiendo así que el término $S(w) \cong 0$. Nos encontramos entonces ante el método de Levenberg-Maquardt. Se suele tomar $\mu = 0.01$. En este método el parámetro μ se incrementa o disminuye en cada paso:

- Si $E(w(k+1)) \leq E(w(k))$ entonces el parámetro μ se divide por un determinado factor ß.
- Si ocurre lo contrario entonces se multiplica por el factor ß.

2.4 Redes neuronales de base radial

2.4.1 Introducción

Las redes neuronales de base radial surgen con el objeto de conseguir redes neuronales cuyo proceso de aprendizaje fuera mucho más rápido que en las redes perceptron multicapa. Para ello se diseñó una red en la que las neuronas de su única capa oculta tuvieran un carácter local, de tal manera que a la entrada de nuevos patrones hubiera menos neuronas que procesar. Ésta es de hecho su principal diferencia con las redes perceptron multicapa, el comportamiento local de sus neuronas, de tal manera que solo unas pocas se activan ante la entrada de un determinado patrón. Esto le confiere determinadas ventajas e inconvenientes en comparación con las primeras.

De todas maneras, y al igual que sucedía con el perceptron multicapa, es un aproximador universal en el espacio $\mathbb{R}^n$.

2.4.2 Arquitectura

Las redes neuronales de base radial disponen de una capa de neuronas de entrada, otra capa de neuronas de salida y solo una capa de neuronas ocultas.

Las conexiones en este tipo de red son todas hacia adelante, es decir, la entrada de todas las neuronas de la capa oculta procede de la salida de todas las neuronas de la capa de entrada, y a su vez las neuronas de la capa oculta transmiten su salida a todas las neuronas de la capa de salida. Un ejemplo de arquitectura de este tipo de red se muestra en la figura 2.3.

La capa de neuronas de entrada únicamente propagan las entradas hacia la capa oculta, es decir, el peso de sus conexiones es siempre igual a uno. Por otro lado, las conexiones entre las neuronas de la capa oculta y las de la capa de salida si tienen asociado un peso.

Las neuronas de salida solo realizan una combinación lineal de las entradas que reciben, es decir, su función de activación es la identidad. Por ello el único procesamiento no lineal que se da en este tipo de redes tiene lugar en las neuronas de la capa oculta.

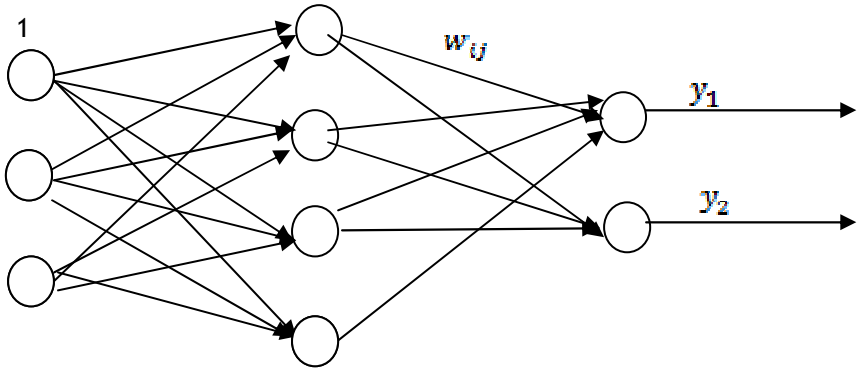

Figura 2.3: Esquema de red neuronal de base radial

Según lo que hemos dicho de cómo se propaga la señal a través de la red la salida y_1 bajo un determinado patrón de entrada $X = (x_1, x_2, x_3)$, donde x_i es la entrada a la neurona i de la capa de entrada, será:

$$y_1 = \sum_{i=1}^{n_{co}} w_{i1}\phi_i(X) + u_1 \tag{2.39}$$

La función $\phi_i(X)$ es la función de activación de la neurona i de la capa oculta. Las neuronas de la capa oculta tienen un comportamiento local debido al tipo de función que es. Esta función tiene la forma siguiente:

$$\phi_i(X) = \phi\left(\frac{\|X - C_i\|}{d_i}\right) \tag{2.40}$$

Es decir, el nivel de activación de la neurona i de la capa oculta depende de la distancia entre el patrón y el centro C_i asociado a dicha neurona. El parámetro d_i mide de alguna forma el campo de actuación de la neurona. De esta manera, si la entrada se encuentra cerca del centro de la neurona el nivel de activación de la misma será grande, y en caso contrario pequeño. Podemos observar como el centro C_i determina la zona de actuación de la neurona y el parámetro d_i la extensión de dicha zona. Por ello se dice que estas neuronas tienen un comportamiento local. Esta función de activación, también conocida como función de base radial puede adoptar diversas formas que se adecúan a este esquema. Algunas de ellas son:

-Función gaussiana $$\phi(r) = e^{\left(-\frac{r^2}{2}\right)} \tag{2.41}$$

-Función inversa cuadrática $$\phi(r) = \frac{1}{1 + r^2} \tag{2.42}$$

-Función inversa multicuadrática $$\phi(r) = \frac{1}{\sqrt{1 + r^2}} \tag{2.43}$$

Donde $r = \frac{\|X - C_i\|}{d_i}$, es decir, la distancia euclídea entre la entrada X y el centro de la neurona dividido por la amplitud. Hay que tener en cuenta que el centro tendrá tantas

componentes como entradas tenga la red para situarlo en el espacio $\mathbb{R}^n$, donde n es el número de neuronas de entrada que existen en la red.

Por todo lo explicado anteriormente llegamos a la conclusión de que únicamente hay que decidir (en relación a la arquitectura de la red) el número de neuronas en la capa de entrada, en la de salida y en la oculta. El número de las dos primeras vendrá determinado por la naturaleza del problema al que nos enfrentamos, es decir, el número de neuronas en la capa de entrada viene determinado por las entradas necesarias para la resolución del problema, y el número en la capa de salida por el número de salidas diferentes que queramos obtener. Sin embargo la determinación del número óptimo de neuronas en la capa oculta es más difícil de determinar. Hay que tener en cuenta que si existen pocas neuronas, se puede representar mal el espacio del problema, y por el contrario, si existen demasiadas neuronas, pueden existir varias neuronas en una determinada región, lo que conlleva consecuencias negativas en la aptitud de la red para resolver el problema. Este problema se suele resolver mediante el procedimiento ensayo-error añadiendo cada vez más neuronas hasta obtener un resultado óptimo.

2.4.3 Entrenamiento

Como ocurría con la red anterior, en esta fase se adaptaban los parámetros de la red para que ésta fuera capaz obtener las salidas apropiadas ante las entradas que se le iban a presentar. En el caso de las redes neuronales de base radial hay que determinar los centros y amplitudes de actuación de cada neurona, así como los pesos de las conexiones y valores umbrales de las neuronas de la capa de salida. Existen dos políticas a la hora de determinar estos parámetros. En la primera de ellas, llamada método híbrido, los centros y amplitudes de cada neurona se determinan mediante un esquema de aprendizaje no supervisado, y los pesos y valores umbrales se determinan mediante un esquema de aprendizaje supervisado minimizando el error que comete la red ante unos determinados patrones de entrenamiento. En el segundo método, denominado método de aprendizaje totalmente supervisado, se determinan todos los parámetros de la red mediante la minimización del error cometido por la red ante unos determinados patrones.

-Método híbrido

Como se comentó anteriormente, en este método primero se determinan los centros y amplitudes de cada neurona mediante un esquema de aprendizaje no supervisado, y posteriormente se determinan los pesos y valores umbrales de las neuronas de salida mediante un esquema de aprendizaje supervisado.

En la determinación de los centros se emplean algoritmos que tienen como objetivo clasificar de manera óptima, en un determinado número de clases, el espacio de entrada definido por los patrones de entrada. Uno de estos algoritmos es el algoritmo de K-medias.

Este algoritmo divide el espacio de entrada en K clases, siendo K el número de neuronas en la capa oculta, de tal manera que se minimice la suma de las distancias de cada entrada a su centro más cercano, es decir, se intenta resolver el problema:

$$\min Z = \sum_{i=1}^{K} \sum_{n=1}^{N} Min \|X(n) - C_i\| \qquad (2.44)$$

Para resolver este problema, este algoritmo propone seguir unos determinados pasos.

1. Se inicializan los centros de las K clases de manera aleatoria, o con unos determinados valores obtenidos por otro método.
2. A cada entrada del patrón se le asocia el centro más cercano. Es decir, el patrón $X(n)$ pertenece al centro C_i si se cumple $\|X(n) - C_i\| \leq \|X(n) - C_s\|$ donde C_s es el centro más cercano encontrado hasta ahora y distinto del centro C_i.
3. Se determinan los nuevos centros de cada clase mediante la media de las entradas que pertenecen a dicha clase.
4. Se repite los pasos 2 y 3 hasta que la posición de los centros no se modifique bajo un determinado margen de tolerancia, es decir que $\|C_i^{nuevo} - C_i^{anterior}\| \leq \varepsilon$ para cualquier i.

Este algoritmo converge muy rápidamente, pero tiene el inconveniente de que se pueden obtener diferentes resultados dependiendo de los valores tomados en la inicialización. Esto ocurre porque el método encuentra mínimos locales de la función Z.

Para la determinación de las amplitudes d_i se calcula de forma que el solapamiento entre zonas de activación de neuronas ocultas se lo menor posible, de manera que se obtengan interpolaciones más suaves. Para ello se emplean diversos heurísticos entre los que destacan los siguientes:

- Media uniforme de la distancia a los p centros más cercanos. Es decir,

$$d_i = \frac{1}{p} \sum_{p} \|C_i - C_p\| \qquad (2.45)$$

- Media geométrica de la distancia a sus dos centros más cercanos, es decir,

$$d_i = \sqrt{\|C_i - C_s\| \|C_i - C_t\|} \qquad (2.46)$$

Por último, hay que determinar los pesos de las conexiones de la red. Para ello se pasa a la fase supervisada del método híbrido. Al igual como ocurría en las redes perceptron consiste en resolver el problema de minimizar una determinada función error. En el caso general esta función error vendrá definida por la expresión dada en la ecuación 2.6. Como ocurría anteriormente, en este método la actualización de los pesos se lleva a cabo mediante la expresión:

$$w_{ik}(n) = w_{ik}(n-1) - \alpha \frac{\partial e(n)}{\partial w_{ik}} \tag{2.47}$$

$$u_k(n) = u_k(n-1) - \alpha \frac{\partial e(n)}{\partial u_k} \tag{2.48}$$

Teniendo en cuenta la expresión del error y que el peso asociado a la conexión entre la neurona oculta i y la neurona de salida k solo afecta a la salida de la neurona k, tenemos,

$$\frac{\partial e(n)}{\partial w_{ik}} = -(s_k(n) - y_k(n)) \frac{\partial y_k(n)}{\partial w_{ik}} \tag{2.49}$$

$$\frac{\partial e(n)}{\partial u_k} = -(s_k(n) - y_k(n)) \frac{\partial y_k(n)}{\partial u_k} \tag{2.50}$$

Además la salida de la neurona k es una combinación lineal de las salidas de las neuronas ocultas ponderada mediante el peso de las respectivas conexiones más el valor umbral, por tanto,

$$\frac{\partial y_k(n)}{\partial w_{ik}} = \phi_i(n) \tag{2.51}$$

$$\frac{\partial y_k(n)}{\partial u_k} = 1 \tag{2.52}$$

De esta manera queda que la actualización de los pesos y valores umbrales dadas por las ecuaciones 2.47 y 2.48 quedan definidos de la forma:

$$w_{ik}(n) = w_{ik}(n-1) + \alpha\big(s_k(n) - y_k(n)\big)\phi_i(n) \tag{2.53}$$

$$u_k(n) = u_k(n-1) + \alpha(s_k(n) - y_k(n)) \tag{2.54}$$

Debido a que la salida de la red depende linealmente de los pesos y valores umbrales, se puede obtener la solución del problema de minimización del error por medio del empleo del método de los mínimos cuadrados. En este método se obtiene directamente la matriz de pesos y valores umbrales W por medio de la expresión:

$$W = (G^t G)^{-1} G^t \cdot S \tag{2.55}$$

Donde la matriz G viene definida por:

$$G = \begin{pmatrix} \phi_1(1) & \phi_2(1) & \cdots & \phi_m(1) & 1 \\ \phi_1(2) & \phi_2(2) & \cdots & \phi_m(2) & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \phi_1(N) & \phi_2(N) & \cdots & \phi_m(N) & 1 \end{pmatrix} \quad (2.56)$$

Donde $\phi_i(n)$ es el valor de la función de activación de la neurona i ante el patrón $X(n)$. Y la matriz S viene definida por:

$$S = \begin{pmatrix} s_1(1) & s_2(1) & \cdots & s_r(1) \\ s_1(2) & s_2(2) & \cdots & s_r(2) \\ \cdots & \cdots & \cdots & \cdots \\ s_1(N) & s_2(N) & \cdots & s_r(N) \end{pmatrix} \quad (2.57)$$

Donde $s_k(n)$ es la salida de la neurona k bajo la entrada a la red del patrón $X(n)$.

- ***Método de aprendizaje totalmente supervisado***

Como ya se comentó, en el aprendizaje totalmente supervisado se determinan los centros, amplitudes y pesos de tal manera que se minimice el error cometido por la red, por lo que no es de esperar que se conserve el comportamiento local de la misma, ya que no se restringe el solapamiento de las regiones de activación de las neuronas.

Para ello hay que tener en cuenta que la dependencia de la salida de la red en relación a los centros y amplitudes de las distintas neuronas de la capa oculta es no lineal, por lo que hay que emplear técnicas de optimización no lineales. A continuación explicaremos el método del descenso por el gradiente. Mediante este método, y como ya se ha repetido en anteriores ocasiones, la actualización de los parámetros (centros, amplitudes y pesos) se lleva a cabo mediante las siguientes expresiones.

$$w_{ik}(n) = w_{ik}(n-1) - \alpha_1 \frac{\partial e(n)}{\partial w_{ik}} \quad (2.58)$$

$$u_k(n) = u_k(n-1) - \alpha_2 \frac{\partial e(n)}{\partial u_k} \quad (2.59)$$

$$c_{ij}(n) = c_{ij}(n-1) - \alpha_3 \frac{\partial e(n)}{\partial c_{ij}} \quad (2.60)$$

$$d_i(n) = d_i(n-1) - \alpha_4 \frac{\partial e(n)}{\partial d_i} \quad (2.61)$$

Donde $j = 1,2,\ldots,p$, $i = 1,2,\ldots,m$, y $k = 1,2,\ldots,r$, siendo p el número de neuronas en la capa de entrada, m el número de neuronas en la capa oculta y r el número de neuronas de salida.

Pesos y umbrales: Ya se determinó en el apartado anterior. La actualización de los pesos y umbrales vienen dadas por las ecuaciones 2.53 y 2.54 respectivamente.

Centros: Teniendo en cuenta que el centro c_{ij} de la neurona i de la capa oculta afecta a todas las salidas de la red, se tiene, aplicando la regla de la cadena:

$$\frac{\partial e(n)}{\partial c_{ij}} = -\sum_{k=1}^{r}(s_k(n) - y_k(n))\frac{\partial y_k(n)}{\partial c_{ij}} \tag{2.62}$$

Por otro lado, el centro c_{ij} solo afecta a la neurona i de la capa oculta, es decir, en la salida solo afecta a la entrada de cada neurona de salida procedente de la neurona i de la capa oculta. Por tanto, se tiene:

$$\frac{\partial y_k(n)}{\partial c_{ij}} = w_{ik}\frac{\partial \phi_i(n)}{\partial c_{ij}} \tag{2.63}$$

Además, teniendo en cuenta la ecuación 2.40 para la función ϕ, tenemos:

$$\frac{\partial \phi_i(n)}{\partial c_{ij}} = \phi_i(n)\frac{x_j - c_{ij}}{{d_i}^2} \tag{2.64}$$

De esta manera, y mediante sustitución en la ecuación 2. 60 obtenemos la ley de actualización de los centros c_{ij}.

$$c_{ij}(n) = c_{ij}(n-1) + \alpha_2(\sum_{k=1}^{r}(s_k(n) - y_k(n))w_{ik})\phi_i(n)\frac{x_j - c_{ij}(n-1)}{{d_i}^2(n-1)} \tag{2.65}$$

Amplitudes: Al igual que ocurría con los centros, la amplitud d_i afecta a todas las salidas de la red, por lo tanto:

$$\frac{\partial e(n)}{\partial d_i} = -\sum_{k=1}^{r}(s_k(n) - y_k(n))\frac{\partial y_k(n)}{\partial d_i} \tag{2.66}$$

Además, la amplitud d_i solo afecta a la entrada de cada neurona de salida procedente de la neurona i de la capa oculta, al igual que ocurre en el caso de los centros. Por tanto,

$$\frac{\partial y_k(n)}{\partial d_i} = w_{ik}\frac{\partial \phi_i(n)}{\partial d_i} \tag{2.67}$$

Y atendiendo a la expresión dada en la ecuación 2.40 tenemos que:

$$\frac{\partial \phi_i(n)}{\partial d_i} = \phi_i(n) \frac{\|X(n) - C_i\|^2}{{d_i}^3} \tag{2.68}$$

De esta manera, y mediante sustitución, se obtiene la ley de actualización para las amplitudes utilizando la ecuación 2.61

$$d_i(n) = d_i(n-1) + \alpha_4 \left(\sum_{k=1}^{r} \left(s_k(n) - y_k(n)\right) w_{ik} \phi_i(n) \frac{\|X(n) - C_i(n-1)\|^2}{{d_i}^3(n-1)}\right) \tag{2.69}$$

Como en todo proceso iterativo hay que inicializar todos los parámetros. Los centros y amplitudes se deberían inicializar con valores que permitan una buena representación del espacio donde se encuentran los patrones de entrada.

-Combinación de los dos métodos

El método híbrido tiene la ventaja de que conserva el comportamiento propio de las redes neuronales de base radial, sin embargo su respuesta es menos exacta que la de las redes obtenidas por el método totalmente supervisado. Por ello, se busca una combinación de los dos métodos para obtener las ventajas de los dos tipos de entrenamiento. De esta manera, en este tipo de entrenamiento, se emplea el algoritmo de K-medias y la determinación de las amplitudes utilizada en el método híbrido, para introducir después estos valores en el entrenamiento totalmente supervisado como valores iniciales. Así se consigue las propiedades locales del primer tipo de entrenamiento, y la exactitud del aprendizaje totalmente supervisado.

CAPÍTULO 3: Redes neuronales en Matlab

3.1. Redes Perceptron Multicapa

3.1.1 Creación de la red

En Matlab se crea una red perceptron multicapa con conexiones hacia delante utilizando la función *newff*. Esta función devuelve una variable que representa a la red. Los argumentos de entrada de esta función (descritos en el orden en el que se colocan) son:

- *R* que es una matriz cuyo número de filas es el número de entradas, y su número de columnas es igual a 2. Esta matriz debe señalar los valores mínimos y máximos que pueden tomar cada una de las entradas a la red.
- *[Slayer1, Slayer2,…]* es un vector que describe el número de neuronas que tienen todas las capas ocultas de la red y el número de neuronas de salida. Asimismo el tamaño de dicho vector permite conocer al programa el número de capas que debe tener la red que se va a crear.
- *{Funciones}* es un vector de varias cadenas de caracteres en el que se señala las funciones de activación que va a poseer todas las neuronas de una capa. Así, la primera cadena señalará la función de activación de las neuronas de la primera capa oculta, la segunda señalará la función de las neuronas de la segunda capa, y así sucesivamente. Por ello, este vector debe tener tantas cadenas como capas vaya a tener nuestra red. Para que la función de activación sea lineal, la cadena debe ser *'purelin'*, para que ésta sea una función tangente hiperbólica se debe escribir *'tansig'*, y para indicar que queremos una función sigmoidal escribiremos *'logsig'*. Por defecto Matlab utiliza la función tangente hiperbólica.
- En el siguiente argumento se indica el tipo de entrenamiento que va a seguir nuestra red. De los diferentes entrenamientos que tiene Matlab se hablará en el 2.1.3. Si no se indica nada el entrenamiento que seguirá es el *trainlm* (ver apartado 2.1.3).

3.1.2 Simulación de la red

Para obtener las salidas de una determinada red ante unas ciertas entradas se utiliza la función *sim*. Esta función devuelve un vector cuyas componentes son las salidas que se obtienen de cada neurona de salida de la red. A dicha función se le introducen como argumentos el nombre del objeto que representa a la red en Matlab, y el vector de entradas a la red.

Se pueden obtener varias simulaciones de un tiempo introduciendo una matriz, cuyos vectores sea cada uno de los vectores de entrada de los que se quiere obtener su salida. De

esta manera, si se tiene una red *"net"* de la que se quiere obtener su respuesta ante unas entradas *"a"*, se escribe en Matlab la siguiente línea de código.

p=sim (net,a);

3.1.3 Entrenamiento

Como se ha comentado anteriormente, el entrenamiento de la red (el que se utiliza en este proyecto) consiste en presentarle unas entradas y sus correspondientes salidas (que son conocidas por nosotros), para que la red vaya reajustando su salida mediante la modificación de sus pesos y valores umbrales, de manera que el error de actuación de la red se minimice. Dicho valor Matlab lo almacena en la variable *net.performFcn*. La medida del error por defecto en Matlab es el error medio cuadrático, que viene dado por la expresión:

Para indicar que se quiere utilizar esta definición del error en Matlab, la variable *net.performFcn* debe igualarse a *mse* (en cualquier caso, este valor lo toma por defecto).

$$E(n) = \frac{1}{N}\sum_{k=1}^{N}\sum_{n=1}^{r}(s_k(n) - y_k(n))^2 \qquad (3.1)$$

Para entrenar una red en Matlab se utiliza la función *train*. Dicha función utiliza como argumentos el nombre de la red que se quiere entrenar (y que anteriormente ha debido ser creada) y los patrones, compuestos por unos vectores de entradas (unidos en una matriz *"a"*) y sus correspondientes salidas (todas ellas en una matriz *"p"*). De esta manera la línea de código que se debería escribir es:

[net,pr] = train (net,a,p);

La variable *pr* contiene información sobre el proceso de entrenamiento y la variable *net* contiene a la red ya entrenada, es decir con sus pesos y valores umbrales ajustados. Por otro lado, destacar que esta función utiliza una serie de variables para definir el entrenamiento y que pueden definirse con anterioridad. Estas variables varían según el tipo de entrenamiento utilizado por lo que se hablará de ellas cuando se describan como se definen los diferentes algoritmos de aprendizaje en Matlab.

Se pueden utilizar diferentes tipos de entrenamiento para las redes del tipo multipercpetrón. A continuación se describirán algunas de ellas.

-***Regla delta generalizada***

Para una descripción más detallada del algoritmo mirar apartado 1.3.3. Resumiendo, dicho algoritmo se basa en actualizar los pesos y valores umbrales en la dirección del gradiente de la función del error. Existen dos formas de actualizar los pesos, se puede realizar en el modo conocido en Matlab como *incremental mode,* bajo el cual los pesos son actualizados en cada entrada de un patrón, o utilizar el *Batch mode* bajo el cual estos no son actualizados hasta que todos los patrones no se hayan introducido en la red. Es decir, solo se actualizan cuando se termina una época de entrenamiento. A no ser que se indique lo contrario se utilizará este último modo. Para ejecutar el entrenamiento en *Batch mode* es necesario introducir los patrones en forma de matriz.

Para utilizar este entrenamiento básico (regla delta generalizada), se introduce en el argumento correspondiente al entrenamiento (mirar apartado 3.1.1) la cadena *'traingd'*. Existen siete parámetros asociados con este tipo de entrenamiento. Éstos son:

- *ephocs*: Define el máximo número de épocas de entrenamiento que puede tener nuestro proceso de aprendizaje (ver apartado 2.3.3).
- *show*: Indica a Matlab la forma de visualización que deseamos tener durante el entrenamiento de la red. Si su valor es *Nan* quiere decir que no se quiere ningún tipo de visualización.
- *goal*: Este variable indica un valor mínimo límite que puede alcanzar la función error de la red. Si ésta alcanza dicho valor el entrenamiento se parará automáticamente.
- *time*: Este parámetro indica el tiempo máximo en segundos que puede durar el entrenamiento de la red. Una vez que el tiempo del proceso alcance dicho valor el entrenamiento se detendrá.
- *min_grad*: Determina el valor mínimo necesario que debe tener el gradiente para detener el algoritmo.
- *max_fail*: Es el máximo número de iteraciones que puede incrementarse el error de validación antes de detenerse el entrenamiento. Ver siguiente apartado
- *lr*: Es el ratio de aprendizaje α. Para más información ir al apartado 1.3.3.

Para llamar a estos parámetros se utiliza la siguiente línea de código (en el caso de que se llame a la variable *ephocs*, para cualquier otro parámetro se actuaría del mismo modo):

```
net.trainParam.ephocs = 50;
```

- ***Regla delta generalizada con la introducción de un momentum:***

Este algoritmo es una variante del anterior. En él se introduce un momento que le permite aminorar los saltos cuando se está acercando al mínimo e ignorar mínimos locales. Para más detalles ir al apartado 2.3.4.

Para señalar que queremos este tipo de entrenamiento hay que introducir la cadena *traingdm* en el lugar apropiado. El único parámetro nuevo con respecto al anterior método es el momento. Éste está almacenado en la variable *mc*. Para acceder a esta variable se procede de la misma forma que con los anteriores métodos.

Los siguientes entrenamientos que se describen poseen un mayor orden de convergencia que los dos anteriores. Se dividen en dos categorías, los que utilizan técnicas heurísticas y los que utilizan técnicas de optimización.

- ***Ratio de aprendizaje variable***

Para ver una descripción más detallada del algoritmo mirar el apartado 2.3.4. En resumen, este algoritmo va variando el ratio de aprendizaje para que el orden de convergencia sea mayor.

La cadena que hay que introducir para implementar este tipo de entrenamiento es *traingda*. Las nuevas variables que hay que controlar son, por un lado, el factor de amplificación máximo que puede tener el error entre una época y la siguiente, que se almacena en la variable *max_perf_inc*, el factor de reducción que en su caso se aplique al ratio de aprendizaje, que se almacena en *lr_dec*, y el factor de amplificación que se le aplica cuando el error obtenido es menor que en la iteración anterior, que se encuentra almacenado en la variable *lr_inc*.

Existe otra función que implementa un entrenamiento similar a *traingda* pero que incluye un momentum. Es invocado con la cadena *traingdx*. La única salvedad con el último es que hay que tener en cuenta la variable *mc* que almacena el valor del momentum.

- ***Método elástico de retropropagación***

Como se comenta en el apartado 2.3.4, este algoritmo se utiliza cuando la red tiene funciones de activación sigmoideas, para evitar la lenta convergencia que se puede dar en este tipo de redes cuando las entradas a las neuronas son muy grandes. Para ello, no utiliza el valor del gradiente para la actualización de pesos, sino que la magnitud del cambio viene definido por otra variable. Para ello Matlab define *delt_inc* para mayorar la magnitud del cambio si los signos de las derivadas coinciden en dos iteraciones consecutivas, *delt_dec* para

minorarlo cuando esto no sucede. Además también se definen la magnitud inicial del cambio mediante la variable *delta0* y el valor máximo que puede tomar dicha magnitud, que se almacena en la variable *deltamax*. Para implementar este tipo de entrenamiento se utiliza la cadena *trainrp*.

-***Métodos de orden de convergencia mayor***

Los algoritmos anteriores, basados en la regla delta generalizada se basan en cambiar los pesos en la dirección de máximo crecimiento del error. Pero ésta no tiene por qué ser la dirección por donde más rápido se converge. En esta idea se basan estos algoritmos.

Por un lado tenemos la actualización de Fletcher Reeves y por otro, tenemos el algoritmo de Newton. Todos estos algoritmos utilizan demasiada memoria, ya que para calcular la dirección de actualización utilizan la matriz hessiana que contiene derivadas segundas, por lo que en la práctica se utiliza aproximaciones de éstos.

Así tenemos los algoritmos de cuasi-Newton que no calculan las derivadas segundas sino que utilizan la secante. Para más información ir al apartado 2.3.4. Por último, el algoritmo de entrenamiento que por defecto utiliza Matlab es el algoritmo de Levenberg-Marquardt. Aproxima la matriz hessiana mediante la matriz jacobiana (mirar apartado 2.3.4). Existe en la formulación un parámetro μ. Las nuevas variables a controlar en este algoritmo son *mu_dec*, *mu_inc*, *mu_max, mem_reduc*. Este último parámetro sirve para controlar la memoria usada en el algoritmo. Dicho entrenamiento se implementa mediante la cadena de caracteres *trainlm*.

3.1.4 Métodos que se pueden utilizar en Matlab para mejorar la capacidad de generalización de la red.

Como se comento en el capítulo II, uno de los problemas que ocurre durante el entrenamiento de la red es el sobreaprendizaje, que inhibe la capacidad de generalización de la red.

Uno de los métodos para evitar el sobreaprendizaje es diseñar un entrenamiento con la extensión justa, pero es imposible saber de antemano cual debe ser el tamaño de la muestra para una aplicación específica. Por ello existen dos métodos que se han implementado en Matlab para mejorar la capacidad de generalización de la red. Así tenemos el método de regularización, y el de parada temprana (*early stop)*.

- ***Regularización***

Este método implica modificar la definición de la función error utilizada en el entrenamiento. Normalmente ésta es la suma de los cuadrados de las diferencias entre la entrada patrón y la obtenida. Esta sección explica cómo puede ser modificada la función error, y la siguiente describe una rutina que obtiene la definición óptima de la función error para obtener buenas propiedades de generalización de la red.

- Función del error modificada.

Normalmente la función error se define como la media de los cuadrados de las diferencias entre las salidas patrón y la salidas obtenidas de la red. Si a esta expresión le añadimos un término consistente en la media de los cuadrados de los pesos de la red y valores umbrales mejoraríamos la generalización de la red.

$$msereg = \gamma \cdot mse + (1-\gamma)msw \tag{3.2}$$

$$msw = \frac{1}{K}\sum_{j=1}^{K} w_j{}^2 \tag{3.3}$$

Donde γ es un parámetro, y mse la función error definida anteriormente. Esta definición de la función error provoca que los pesos de la red obtenidos sean pequeños, lo que fuerza una respuesta más suave de la red y hace menos probable el fenómeno de sobreaprendizaje.

En Matlab esto se introduce modificando el valor de la variable net.perforFcn al valor 'msereg'. El problema de este método es conocer el valor óptimo del parámetro γ, si es muy grande puedes provocar sobreaprendizaje, si es demasiado pequeño, la red no se adecúa bien a los datos de aprendizaje. Este valor se modifica mediante la siguiente línea de código:

net.performParam.ratio =0.5;

- ***Parada temprana***

Esta técnica divide los datos disponibles en tres partes. La primera parte es el conjunto de entrenamiento que es usada para computar el gradiente y para la actualización de los pesos y valores umbrales. El segundo subconjunto es el de validación. Normalmente el error de validación es monitorizado durante el proceso de entrenamiento. Normalmente este error va decreciendo a medida que transcurre el entrenamiento, pero en el caso de que se produzca sobreaprendizaje, este error comienza a crecer. Cuando el error de validación

comienza a crecer en un especificado número de iteraciones (en la variable *net.trainParam.max_fail*), se detiene el entrenamiento y se utilizan los pesos y umbrales de la iteración de menor error de validación.

El tercer subconjunto es el de generalización. No es usado durante el entrenamiento pero es útil para comparar distintos modelos. Puede ser útil también graficar este error durante el entrenamiento. Si este error muestra un mínimo en un número de iteraciones significativamente diferente al de error de validación, puede mostrar una pobre división de los datos.

Este método se utiliza conjuntamente con cualquiera de los otros métodos de entrenamiento descritos anteriormente. A continuación se muestra un ejemplo de cómo se implementa este método en Matlab:

```
p = [-1:0.05:1];

t = sin(2*pi*p)+0.1*randn(size(p));

val.P = [-0.975:.05:0.975];

val.T = sin(2*pi*v.P)+0.1*randn(size(v.P));

net=newff([-1 1],[20,1],{'tansig','purelin'},'traingdx');

net.trainParam.show = 25;

net.trainParam.epochs = 300;

net = init(net);

[net,tr]=train(net,p,t,[],[],val);
```

La única diferencia es que se ha separado los patrones en patrones de entrenamiento y patrones de validación aleatoriamente, en las cuatro primeras líneas de código.

3.1.5.- Preproceso y postproceso

El entrenamiento de la red puede ser más eficiente si se realiza una serie de transformaciones en las entradas y salidas de la red. A continuación se describe las rutinas de preproceso que se pueden usar.

- Min y Max. Mapminmax

Antes del entrenamiento se suele escalar las entradas y salidas para que siempre pertenezcan a un determinado rango. El siguiente código muestra como se utiliza esta función.

[pn,ps] = mapminmax(p);

[tn,ts] = mapminmax(t);

net = train(net,pn,tn);

Las entradas y salidas originales de la red se encuentran en las matrices "p" y "t" respectivamente. Las entradas y salidas normalizadas se obtienen de las variables *pn* y *tn*. Las variables *ps* y *ts* contienen los valores máximos y mínimos de las entradas y salidas originales respectivamente. Cuando la red ya ha sido entrenada, la variable *ps* puede ser utilizada para transformar las futuras entradas a la red. Y lo mismo para volver a transformar las salidas, pero utilizando la variable *ts*.

A continuación se muestra un trozo de código para hacer esto último:

an = sim (net,pn);

a=mapminmax('reverse',an,ts);

Y para escalar las nuevas entradas de la red se utiliza el siguiente código:

pnew = mapminmax('apply',pnew,ps);

anew =sim(net,pnew);

anew =mapminmax('reverse',anew,ts);

-Función mapstd

Otra manera de escalar las entradas y salidas de la red es normalizarlo para que la media de los valores sea cero y tengan una desviación estándar igual a la unidad. Para ello se utiliza la función *mapstd*, tal y como se muestra a continuación:

[pn,ps] = mapstd (p);

[tn,ts] = mapstd(t);

En este caso *ps* y *ts* contienen la media y desviación estándar original. Estos valores son utilizados posteriormente para escalar las nuevas entradas y salidas de la red, de una

forma similar a como se hizo con la función *mapminmax*. La siguiente sección es una rutina de posproceso útil para analizar el comportamiento de la red.

-Análisis de la red entrenada

A menudo es útil investigar la respuesta de la red con más detalle. Una manera es realizar un análisis de regresión entre la respuesta de la red y sus salidas patrón. Esto se hace con ayuda de la función *postreg*.

A la función *postreg* hay que introducirle la salida de la red y la salida esperada de la misma. Ésta te devuelve tres valores, los dos primeros corresponden a la pendiente y a la ordenada en el origen de la recta de regresión, y el tercero al grado de correlación de los datos. Si el grado de correlación de datos es igual a la unidad quiere decir que la respuesta de la red ante esas entradas es perfecta.

3.2 Redes neuronales de base radial

3.2.1 Introducción

Las redes neuronales de base radial podrían requerir más neuronas que las redes neuronales estándar de retropropagación. Pero su entrenamiento es similar a éstas. Trabajan mejor cuando hay muchos datos de entrenamiento disponibles.

3.2.2 Creación de red

Las redes neuronales de base radial pueden ser implementadas con cualquiera de estas dos funciones, *newrbe* y *newrb*.

- Función newrbe

Esta función coge una matriz de vectores de entrada y sus correspondientes salidas patrón y una constante *SPREAD*, que señala el campo de actuación de cada neurona, y devuelve una red cuyos pesos y valores umbrales son tal, que devuelve exactamente las salidas esperadas P para las entradas A. Su sintaxis en Matlab es la siguiente:

net = newrbe(A,P,SPREAD)

Esta función crea una red cuya capa de neuronas de base radial tiene un número de neuronas igual al número de entradas diferentes que se le proporciona a la red durante su

entrenamiento. De esta manera cada neurona de esta capa actúa como un detector de un tipo de entrada en concreto. Como se ha dicho antes la red tiene un error cero para los valores de entrada.

La constante SPREAD es muy importante porque definirá el campo de actuación de cada neurona. Por ejemplo si dicha constante tiene un valor cuatro, la neurona responderá con un valor 0.5 a cualquier entrada que se encuentre a una distancia euclidea de cuatro con respecto a su centro de actuación. Debe ser lo suficientemente grande para que las neuronas actúen correctamente en regiones de solapamiento. Ello provoca que la respuesta de la red sea muy suave y que tenga buena capacidad de generalización. El problema estriba, en que si se necesitan demasiados vectores de entrada para caracterizar correctamente la red, ésta tendrá en su capa oculta demasiadas neuronas.

- Función newrb

Esta función se puede emplear de otra forma para generar una red más eficiente. En este método, la función crea la red de forma iterativa, creando una neurona más en cada iteración. Las neuronas son añadidas hasta que el error medio cuadrático caiga por debajo del parámetro *GOAL*, o se haya alcanzado un determinado número de neuronas máximo. La sintaxis de dicha función en Matlab es:

net = newrb(P,T,GOAL,SPREAD);

CAPÍTULO 4: Uso de la GUI de Matlab

4.1.-Introducción

La interfaz gráfica de usuario de Matlab (GUI) permite desarrollar ventanas, gráficos, botones, menús, etc. que tienen distintas funcionalidades y además permiten visualizar programas desarrollados en este entorno. La elaboración de ventanas gráficas puede llevarse a cabo de dos formas, por un lado mediante la implementación de un script que defina los distintos elementos de la misma y su funcionalidad, o utilizando una herramienta de diseño de GUIs que posee Matlab llamada GUIDE. En este capítulo solo nos centraremos en el primer modo.

Una ventana gráfica consta de varios elementos. Así tenemos el menú de interfaz con el usuario, los distintos controles que nos permiten ejecutar y controlar acciones en el programa, y los ejes que nos permiten desplegar gráficas e imágenes.

El flujo del programa en una GUI es diferente al de cualquier función que se crea en Matlab. En un script o función definida en Matlab las órdenes se ejecutan secuencialmente siguiendo unas determinadas condiciones lógicas hasta que se llega al final del programa, pudiendo obtenerse o no un resultado final a la salida. Sin embargo, en una GUI las órdenes no se ejecutan en una secuencia preestablecida, sino que es el usuario el que controla el flujo del programa a través de la interactuación con los controles de la GUI. Este ciclo continúa indefinidamente hasta que se cierra el programa.

4.2 Objetos gráficos en Matlab

4.2.1 Estructura jerárquica de los objetos gráficos en Matlab

El objeto más general en Matlab es la pantalla. Solo puede haber una. En esta pantalla (*screen*) puede haber una o varias ventanas (*figures*). En una ventana encontramos uno o más ejes de coordenadas (*axes*), unos controles que nos permiten interactuar con el programa (*uicontrols*) y los menús (*uimenus*). Por último, los ejes pueden contener cinco objetos gráficos, líneas (*line*), polígonos (*patch*), superficies (*surfaces*), imágenes tipo bitmat (*image*) y texto (*text*). Estos objetos siguen una determinada estructura jerárquica de padres e hijos tal y como se ve en la figura 4.1. Cuando se elimina un elemento, se elimina también todos sus *hijos*.

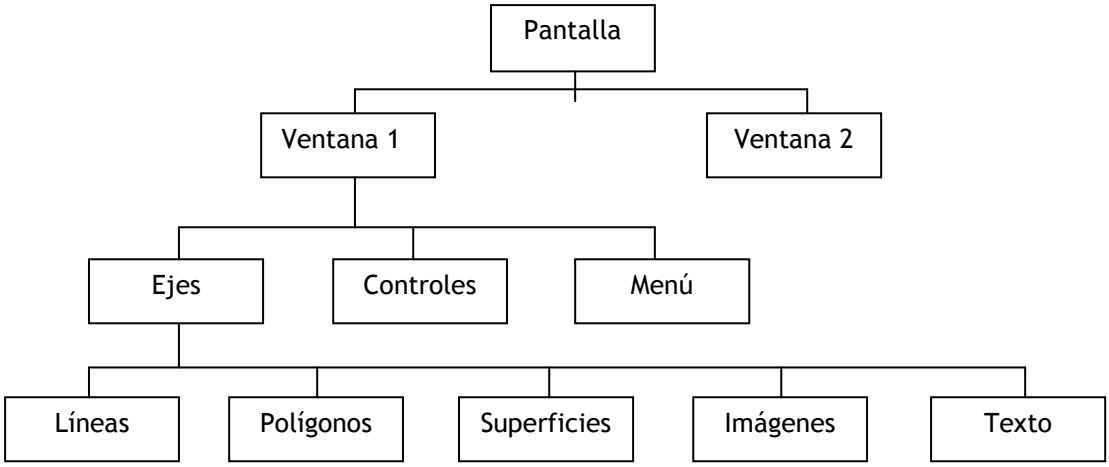

Figura 4.1: Estructura jerárquica de los objetos gráficos en Matlab

4.2.2 Identificadores de los objetos

Todos los objetos gráficos creados en una GUI tienen un identificador o id, también conocido como *handle*. Como se comentó anteriormente solo puede haber una única pantalla, que en Matlab tiene el identificador cero, y puede haber una o varias ventanas que se identifican con números enteros. Los identificadores de otros elementos gráficos son números *float*. Estos identificadores pueden ser obtenidos como valores de retorno y utilizado como variables en el programa.

En Matlab puede haber una o varias ventanas abiertas pero sólo una está activa, y dentro de esta ventana puede haber uno o varios ejes pero sólo uno está activo que es sobre el que se dibujan los elementos. Asimismo puede haber un control activado. Se puede obtener el identificador del objeto activo mediante los siguientes comandos:

- *gcf* devuelve el identificador de la ventana activa.
- *gca* devuelve el identificador de los ejes activos.
- *gco* devuelve el identificador del objeto activo.

El identificador también nos sirve para poder eliminar los objetos mediante el comando *delete(handle)*.

4.2.3 Propiedades de los objetos gráficos.

Todos los elementos gráficos de Matlab tienen una serie de propiedades que los definen. Algunas son comunes a todos los elementos gráficos y otras corresponden al tipo particular al que pertenecen. El listado de propiedades que tiene un objeto se puede obtener mediante el comando *set(handle)*. Este comando solo lista las propiedades que tiene el objeto pero no su valor concreto. Para obtener su valor se utiliza el comando *get(id)*, que lista las mismas propiedades que el anterior pero con su valor actual. Si lo que deseamos es el valor de una o algunas propiedades en concreto también se puede obtener con el comando *get*, pero señalando aquellas que deseamos conocer su valor. Para ello escribimos la siguiente línea de código:

get(handle,'Propiedad');

Asimismo podemos modificar el valor de una propiedad mediante la utilización del comando *set*, especificando el nuevo valor y la propiedad que se va a modificar. De esta manera deberíamos de escribir:

set(handle,'Propiedad',valor);

Las propiedades del objeto se pueden especificar en su creación. En el caso de que el programador no señale nada, ésta toma un valor por defecto. En el caso de que se quiera volver a los valores por defecto de alguna determinada propiedad solo debemos cambiarle dicho valor a *factory*.

4.3. Creación de objetos gráficos

4.3.1 Creación de controles.

Todos los controles se crean con el comando *uicontrol*. En dicha creación es necesario especificar el identificador del padre (si no se señala nada el valor por defecto es el identificador de la ventana que esté activa, es decir *gcf*), las propiedades del objeto que se quieran especificar y las acciones que se deberían ejecutar cuando se active dicho control.

Para implementar todo esto en Matlab se utiliza dicha expresión:

Control =uicontrol(id_padre,…

'Propiedad 1', valor 1,…

'Propiedad 2', valor 2,…

'callback','sentencias');

A continuación se enumeran las propiedades más importantes que dispone el comando *uicontrol*:

- *Tipo de control (Style):* Permite señalar el tipo de control que se desea. Para ello es necesario introducir una cadena concreta (se especificará cuál cuando más adelante se hable de los distintos controles que existen)
- *Control del fondo del objeto (Backgroundcolor):* Controla el color de fondo del objeto. Por omisión suele ser gris claro. Para su modificación es necesario tener en cuenta que los colores se obtienen por la mezcla de los colores rojo, azul y verde mediante el código RGB.
- *Posición del objeto (Position):* Define la posición y el tamaño del objeto dentro del objeto padre. Para ello se define un vector de cuatro elementos. Los primeros dos elementos señalan las coordenadas del vértice inferior izquierdo, y los otros dos señalan el ancho y alto del control.
- *Nombre del objeto (String):* Esta opción define el nombre que aparecerá en el control.
- *Callback:* Señala las acciones que debe ejecutar el programa cuando se activa el control. Pueden ser una serie de sentencias, o la llamada a un archivo .M. Si el archivo .M es una función es necesario que queden especificadas el valor de sus entradas.
- *Identificador del padre:* Señala el objeto padre del control que se está creando. Por defecto es la ventana activa en ese momento.
- *Valor del objeto (Value):* Permite controlar el valor que tiene el control en un momento dado.
- *Control de activado/desactivado (Enable):* Esta opción permite desactivar un control de tal forma que una acción sobre él no produzca ningún efecto. Los valores que puede tomar son *on* y *off*.
- *Visible:* Similar al anterior pero además el control desaparece de la pantalla. Toma los mismos valores que en el caso anterior.

Como se comentó anteriormente, hay distintos tipos de controles bajo los cuales la utilización de alguna de estas propiedades cambia ligeramente. A continuación se describirá la utilidad de los distintos tipos de controles.

- *Texto estático*: La propiedad Style toma el valor *'text'*. Sirve para mostrar mensajes, advertencias e incluso valores numéricos en una GUI. Este control no posee cadena de invocación. El texto a mostrar se especifica en la propiedad *String*. Si se quiere modificar se utiliza el comando *set* como se mencionó anteriormente.
- *Menú desplegable:* Son menús que pueden aparecer en cualquier punto del panel. Es la principal diferencia que tienen con respecto a los menús de interfaz. Las opciones que aparecen en el menú se especifican en la propiedad *String* separadas por barras

(|). Para conocer que opción se ha escogido se debe obtener el valor (*value*) de dicho objeto. La propiedad *Style* toma el valor *'popup'*.

- *Boton:* Son pequeños objetos en la pantalla que al pulsar sobre ellos ejecutan una determinada acción. Generalmente van acompañados por texto, que se especifica en *String*. *Style* toma el valor *'pushbutton'*.
- *Casilla de verificación:* Solo poseen dos valores, encendido o apagado. Para conocer dicho estado es necesario utilizar la función *get(handle, 'value')*. Generalmente se utilizan para ejecutar determinadas acciones sencillas o para indicar ciertas condiciones que afectarán a una determinada acción. Se indican mediante la cadena *'checkbox'*.
- *Botón de radio:* Es un grupo de casillas de verificación que son mutuamente excluyentes entre sí. Se especifica mediante la cadena *'radio'*.
- *Control deslizante:* Es un control que permite modificar un parámetro de manera cuasicontinua. La propiedad *Style* debe ser igual a *'Slider'*.
- *Texto editable:* Permite al usuario introducir una cadena, que posteriormente será leída mediante la función *get* cuando se ejecute dicho control. Se especifica mediante la cadena *'edit'*.
- *Marcos:* Sirve para agrupar distintos dispositivos dentro de una ventana gráfica. Se especifica mediante la cadena *'frame'*.
- *Listas:* Muestra una lista de artículos y permite seleccionar al usuario uno o más artículos. Se indica mediante la cadena *'Listbox'*.

4.3.2 Creación de menús de interfaz con el usuario

Es un menú o grupo de menús que se encuentra en la parte superior de la ventana. Cuando "pinchas" en un menú se te despliega una lista de opciones, que pueden ser acciones propiamente dichas, o un menú más. En Matlab se crean con el comando *uimenu* cuya sintaxis es la siguiente:

m1= uimenu (gcf, 'Label', 'Rotulo1' 'Position' [k] ,'Backgroundcolor', [r,g,b], 'Callback','[]');

La propiedad *Label* especifica el rótulo que aparecerá en el menú. *Position* indica el orden de prioridad del menú, por ello *k* debe ser un número entero. *Callback* solo es necesario si este menú es terminal. En el caso de que se quieran añadir opciones al menú se deben crear más objetos menús, cuyo padre sea éste.

4.3.3. Creación de ejes para la representación de gráficas.

El comando *axes* genera unos ejes (gráfica) en un punto determinado de la ventana. El origen y el tamaño de los ejes se especifican con la propiedad *Position*, que se utiliza del mismo modo que con el comando *uicontrol*, solo que aquí los valores están normalizados. Este comando permite modificar el tamaño y posición de los ejes libremente (ésta es una diferencia importante con respecto al comando *subplot*).

CÁPITULO 5: Descripción de la GUI

5.1. Introducción

En este capítulo se describirá detalladamente la GUI de Redes Neuronales perceptron multicapa y base radial, que ha sido desarrollada en el entorno de Matlab. Se describirá cada una de las funcionalidades que dispone el programa, se realizará una descripción de la misma y se justificará su utilización. Por último se realizará una comparación con otras herramientas existentes que tienen la misma finalidad, en concreto con la GUI que proporciona Matlab para el entrenamiento de redes neuronales, denominada *nntool*. Para iniciar la GUI es necesario escribir en la pantalla principal de Matlab la sentencia Redes_neuronales. Es necesario copiar la carpeta con los scripts de Matlab y las carpetas de ayuda (carpeta Scripts) en el directorio desde donde se ejecuta Matlab. Si se ha realizado correctamente aparece una pantalla de presentación donde se tiene que pulsar el botón entrar.

5.2. Inicio y distribución de los elementos en el programa

La pantalla principal del programa es la que se muestra en la figura 5.1. En ella vemos 4 grandes zonas. La primera zona, en la figura marcada como 1, es la barra de herramientas que contiene todas las funcionalidades de las que dispone el programa. La segunda zona es el panel de errores, que aparece marcada como 2. En este lugar aparecen aquellos parámetros, de la red o del entrenamiento, que el programa detecta que no están bien. La tercera zona,

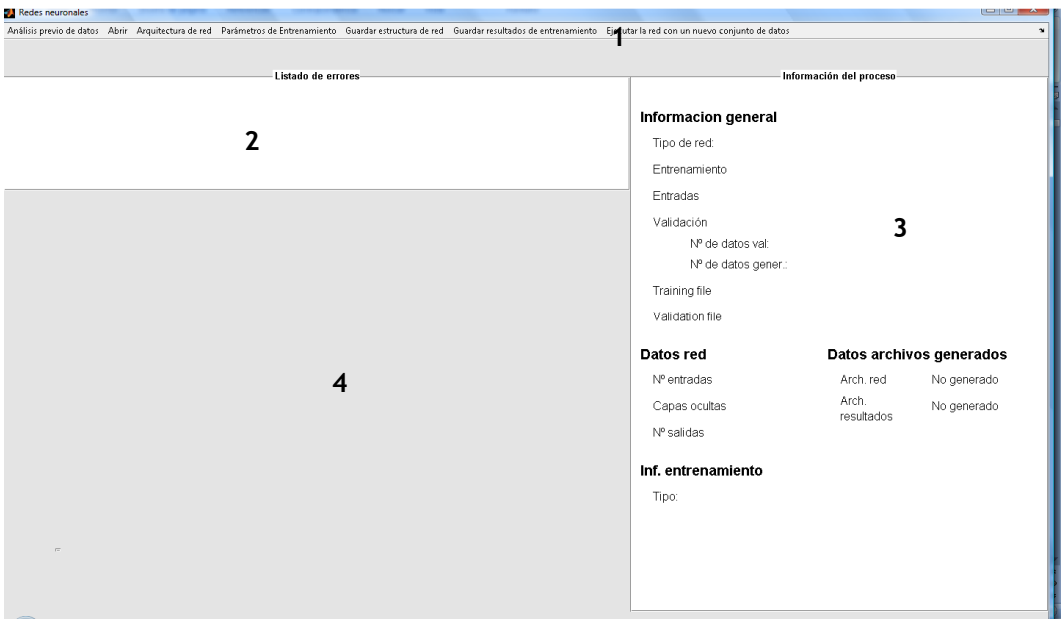

Figura 5.1: Pantalla principal del programa

marcada como 3, es el panel de información del proceso. En ella podremos visualizar el valor de algunos parámetros importantes que se van a utilizar en el entrenamiento. Por último, la cuarta zona, marcada como 4, es la zona donde aparecen todos los menús desplegables, para introducir los valores que nos permitan definir la arquitectura de la red y el entrenamiento, así como los resultados de algunas determinadas funcionalidades.

Selección del Directorio de Trabajo: Nada más arrancar el programa hay una ventana en la zona 4. Dicha ventana permite seleccionar un directorio de trabajo. Dicho directorio es la carpeta donde deben estar todos los ficheros de datos que se vayan a utilizar y donde se va a guardar todos los ficheros que genere el programa. Este directorio no tiene que ser el mismo directorio desde donde se ejecuta Matlab (donde están los ficheros script y de ayuda). Se hablará más detalladamente en el apartado 5.3.2.

5.3.-Funcionalidades del programa.

5.3.1.-Introducción

Como se comentó en el apartado 5.2, se accede a todas las funcionalidades del programa a través de la barra de herramientas. En términos generales, las funcionalidades de esta GUI consisten en ejecutar algún tipo de análisis o introducir el valor de determinados parámetros al programa para poder realizar determinados análisis. Las funciones se agrupan en 9 grandes grupos, que contienen a su vez distintas opciones relacionadas entre sí. Estos 9 grupos son (en la barra de herramientas de izquierda a derecha):

i. *Abrir*
ii. *Análisis previo de datos*
iii. *Arquitectura de red*
iv. *Parámetros de entrenamiento*
v. *Guardar estructura de red*
vi. *Guardar resultados de entrenamiento*
vii. *Ejecutar la red con un nuevo conjunto de dato*
viii. *Postproceso*.
ix. *Ayuda*

A continuación se explicará detalladamente cada uno de estos grupos.

5.3.2.- Abrir.

Dentro de este módulo podemos encontrar las siguientes opciones (tal y como se ve en la figura 5.2): *Red nueva, cargar red y seleccionar directorio.*

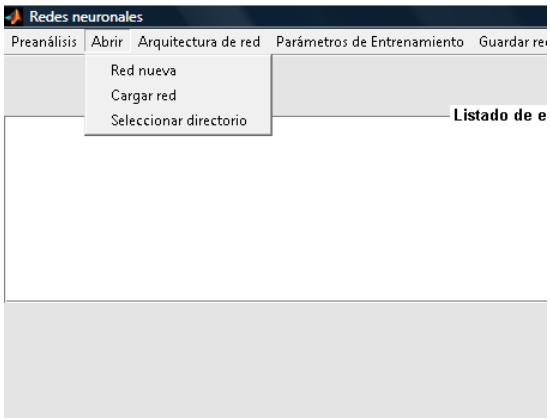

Figura 5.2: Menú desplegable "*Abrir*"

***Red nueva*:** Esta opción permite limpiar todas las variables que maneja el programa y empezar la definición y el entrenamiento de una red nueva. Es muy importante que cuando se vaya a trabajar con un nuevo caso se ejecute esta herramienta.

***Cargar red*:** Esta opción se utiliza para leer una red entrenada que ha sido guardada previamente en *Guardar estructura de red*, opción que se verá en el apartado 5.3.7. Con esta opción se podrá simular redes previamente entrenadas y de esta manera obtener la salida de dichas redes al ejecutarla con nuevos datos.

Cuando se ejecuta esta opción, se abre un panel en la zona 4, tal y como indica la figura 5.3. En dicho panel se solicita el nombre del archivo donde se encuentran los parámetros de la red

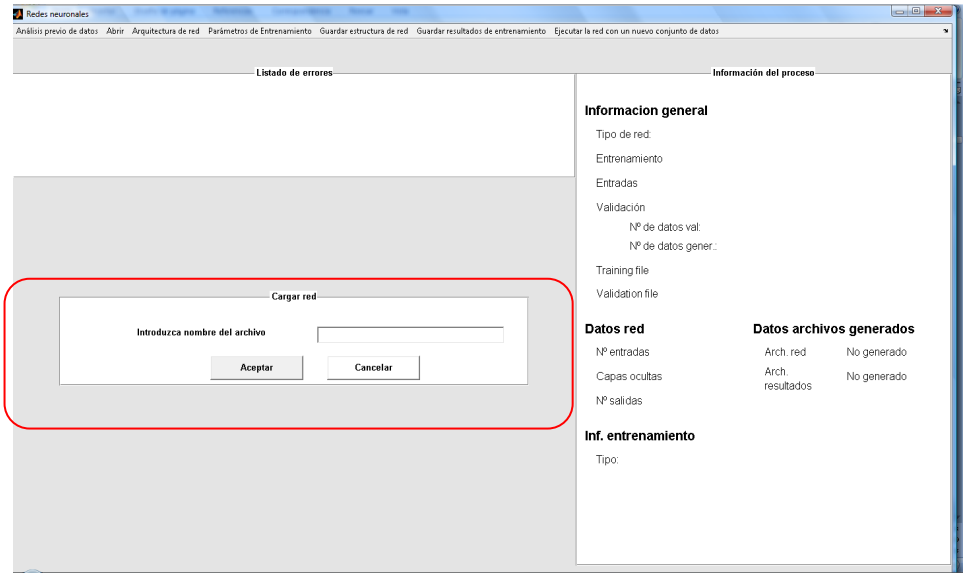

Figura 5.3: Panel para introducir el nombre de la red entrenada a cargar

entrenada. El nombre que hay que indicar es el mismo que se indicó en el momento en que se guardó dicha red. Es importante señalar que para que el programa pueda leer dicho fichero, es importante que éste se encuentre dentro de una carpeta "red_creada", y que a su vez ésta se encuentre en el directorio seleccionado inicialmente.

Una vez que se ha introducido el nombre de la red, se pulsa el botón *aceptar*. Si todo se ha realizado correctamente en la pantalla de información del proceso aparecerá los datos de la arquitectura de la red cargada, tal y como aparece en la figura 5.4. A partir de aquí el programa puede trabajar con nuevos datos utilizando dicha red.

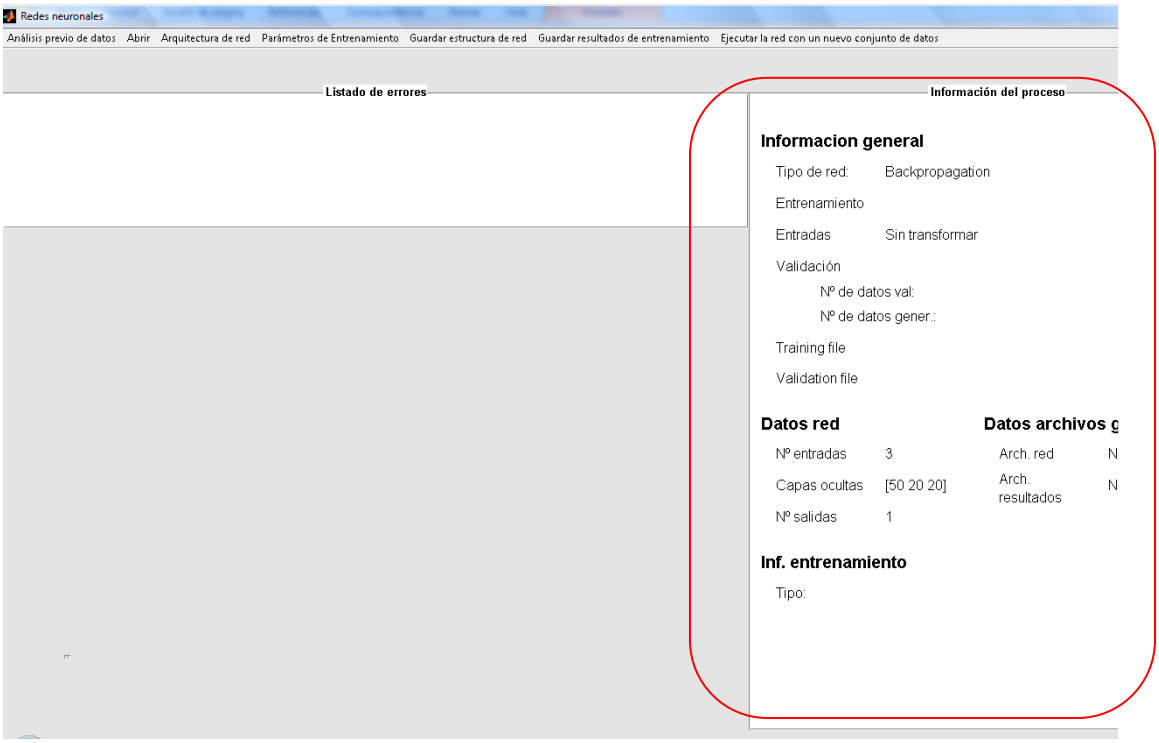

Figura 5.4: Datos de arquitectura de la red cargada en la pantalla información del proceso

***Seleccionar directorio*:** Esta funcionalidad permite elegir el directorio donde el programa guardará todos los archivos que genere. Se ha introducido dicha opción para permitir al usuario poder organizar sus trabajos. Por otro lado el usuario tiene que tener en este directorio todos los archivos de datos que vaya a utilizar con la GUI.

Al pulsar esta opción se abre un panel en la zona 4 (figura 5.5), que solicita introducir la dirección del directorio de trabajo. Una vez indicado se pulsa el botón *aceptar*.

Cabe destacar que cuando arranca el programa, se abre automáticamente este panel para seleccionar el directorio de trabajo. Si no lo hacemos, el programa entiende que el directorio es el que existe en el programa por defecto, es decir C:/ .

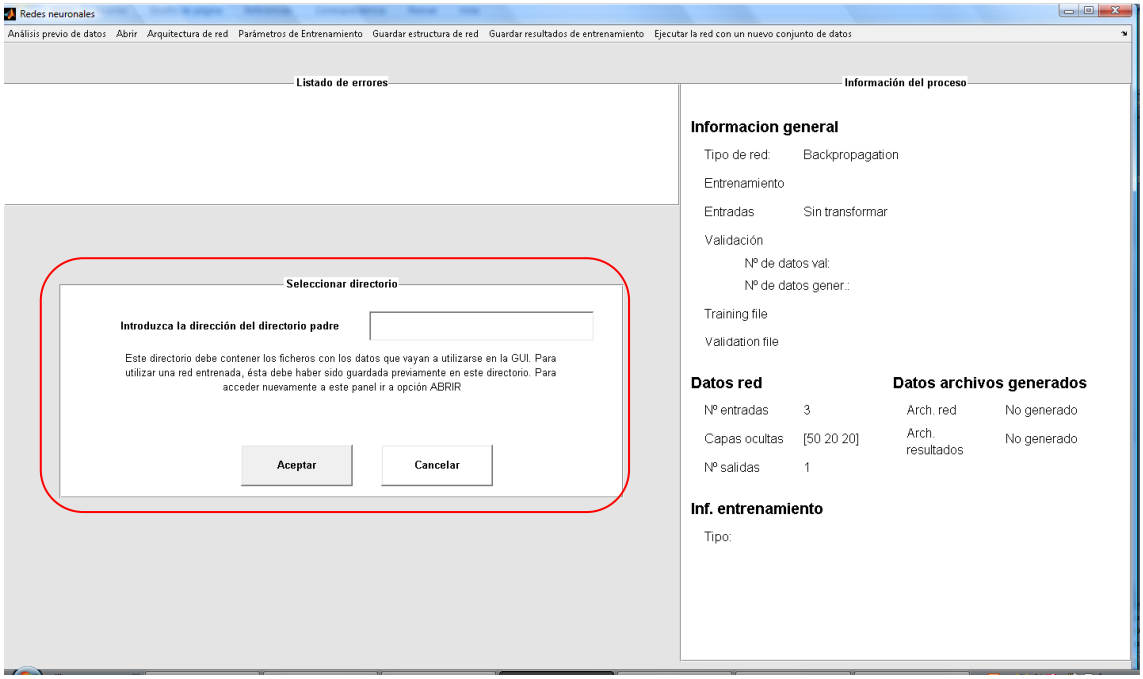

Figura 5.5: Panel para seleccionar el directorio

5.3.3.-Análisis previo de datos.

Este módulo permite realizar un estudio estadístico previo de los datos. Contiene las siguientes opciones (figura 5.6): *Cargar datos*, *Parámetros estadísticos*, *Histograma de las variables*, *Gráfica de Xi vs Yj*, *Boxplot de las entradas y salidas*, *Modelo de regresión lineal múltiple*. En este apartado se explicará cada una de estas opciones.

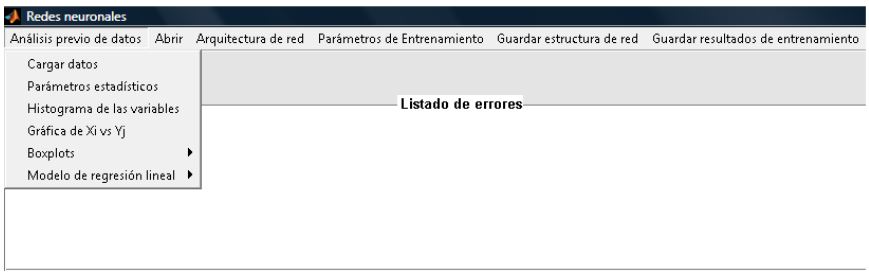

Figura 5.6: Menú Análisis previo de datos

Cargar datos: Esta opción permite cargar el archivo que contiene los datos que se quieren analizar. Para poder cargarlo correctamente el fichero debe estar en el directorio de trabajo. Es importante señalar que para que el programa ejecute las opciones disponibles se debe cargar antes el fichero de los datos. El formato del fichero es *.tex. Las variables de entrada y salida de la red van en columnas, la observación (o dato) por filas. El fichero puede

contener el total de datos, los usados en el entrenamiento y en la generalización de la red, o por el contrario, sólo contener los datos para el entrenamiento.

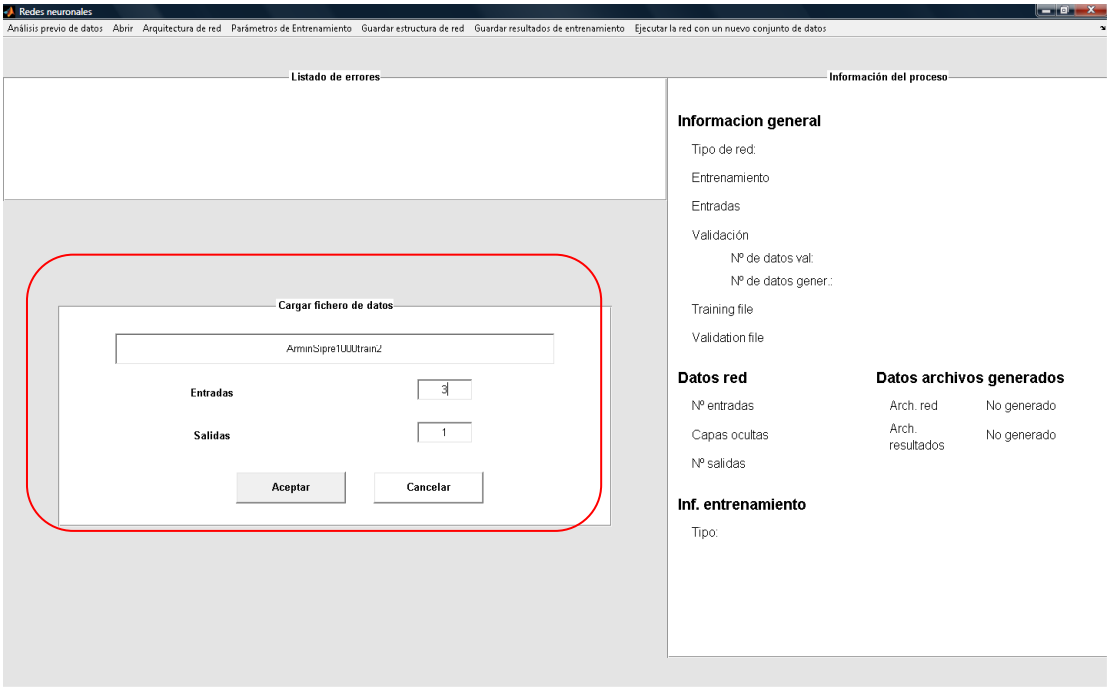

Figura 5.7: Panel para cargar el fichero de datos.

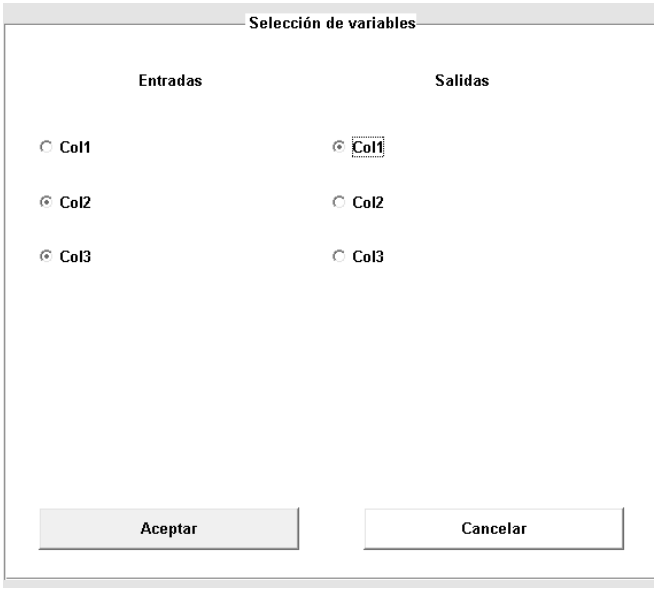

Figura 5.7b: Panel para seleccionar las entradas y salidas

Cuando se accede a esta función se despliega el panel que se muestra en la figura 5.7a. En él hay que especificar el archivo de datos y el número de entradas y salidas. Cuando

se pulse el botón *Aceptar* se despliega un nuevo panel como el de la figura 5.7b. En él hay que especificar qué columnas de datos del fichero son entradas y cuáles son salidas. El número de cada una debe coincidir con el número de entradas y salidas especificados anteriormente.

Parámetros estadísticos: Esta herramienta permite estimar los siguientes parámetros estadísticos de cada una de las variables: media, desviación típica, valor máximo y mínimo, valor del percentil 20 y percentil 80.

La media y la desviación típica muestran al usuario aproximadamente cuál es la distribución de los datos. El máximo, el mínimo y los percentiles permiten al usuario saber si la mayoría de sus datos se concentran alrededor de la media y si la media está centrada en relación a la muestra o no.

Cuando se pulsa la opción *Parámetros estadísticos* aparece un panel con los resultados, como el que se muestra en la figura 5.8. En este panel se denota con *X* a las variables de entrada, y con *Y* las variables salida.

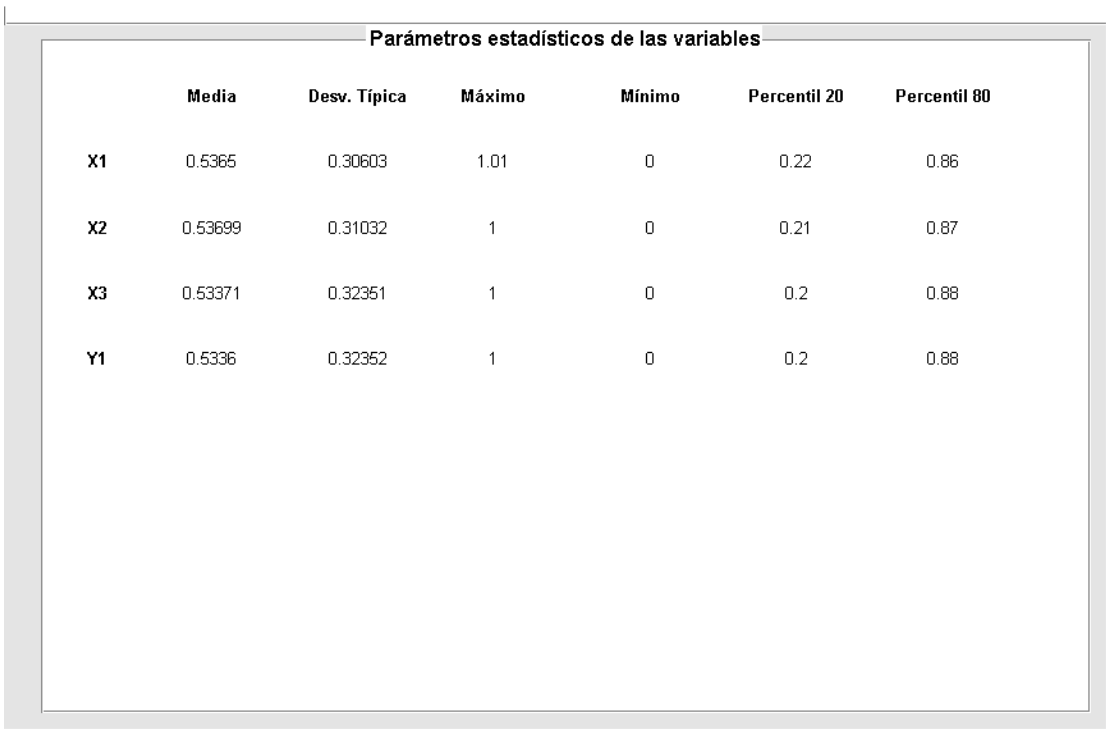

Figura 5.8: Parámetros estadísticos de las variables

Histograma de las variables: Esta herramienta permite visualizar la distribución de cada variable mediante un histograma de frecuencias, como el mostrado en la figura 5.9.

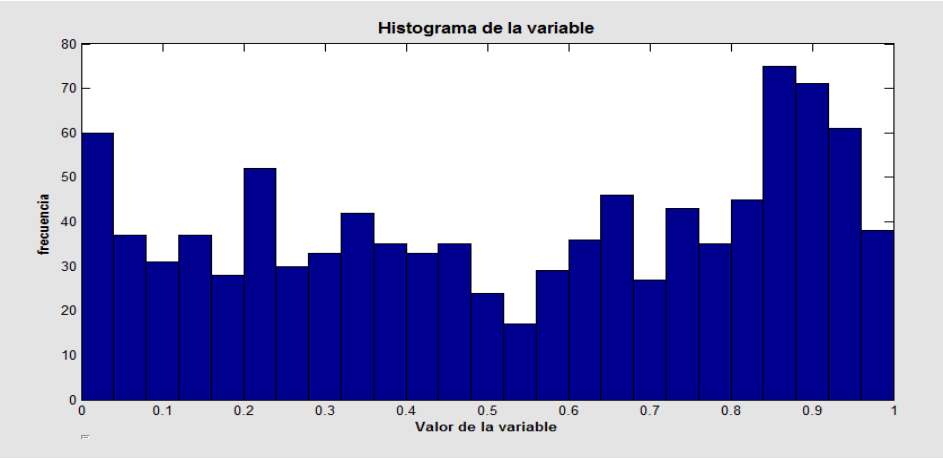

Figura 5.9: Histograma de frecuencias

Cuando se selecciona la herramienta *Histograma de las variables* inicialmente se despliega el panel que se muestra en la figura 5.10. Dicho panel solicita que especifiques de qué variable se quiere obtener el histograma y el número de clases que se desea en el mismo. Una vez que se pulsa el botón *Aceptar* aparece el gráfico representado en la figura 5.9.

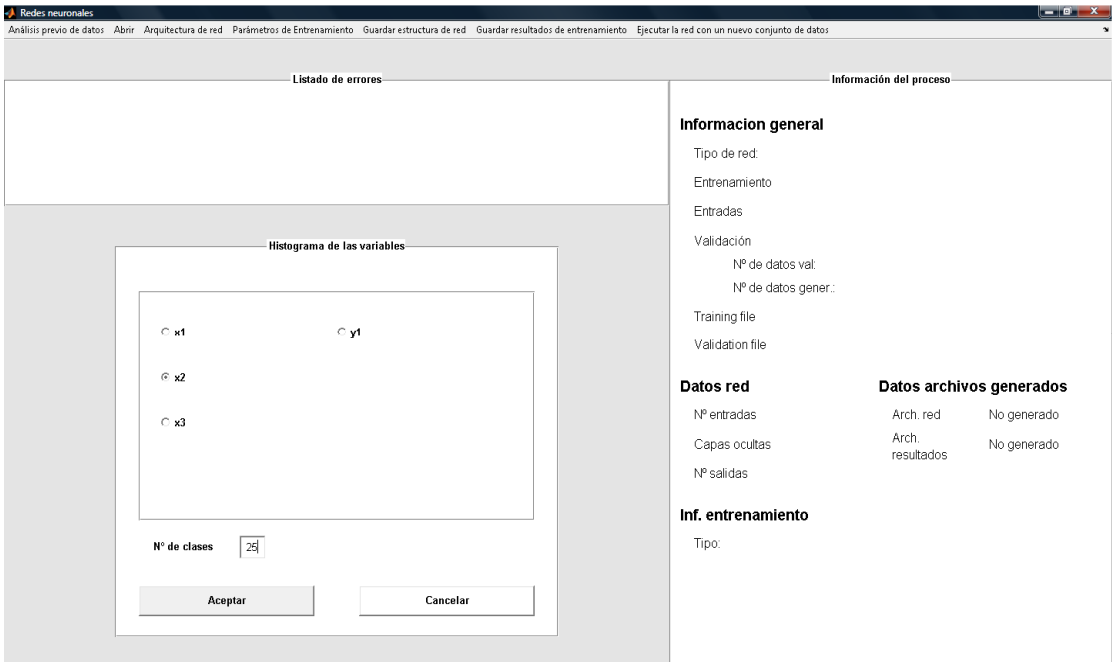

Figura 5.10: Panel para seleccionar las opciones para la elaboración del histograma

Gráfico de Xi vs Yj **:** Esta herramienta muestra un gráfico de dispersión (que muestra la dependencia) entre una variable de entrada frente a una variable de salida.

Cuando se accede a esta herramienta, el programa despliega un panel tal y como se muestra en la figura 5.11. En él hay que especificar que variable de entrada y de salida se utilizará para la construcción de dicho gráfico. Cuando se pulse el botón *Aceptar* el programa mostrará un gráfico como el mostrado en la figura 5.12. El gráfico consiste en una nube de puntos, donde cada punto representa un dato del fichero. Este gráfico ayuda al usuario a visualizar mejor la dependencia de las salidas con las entradas. Por ejemplo, en el caso

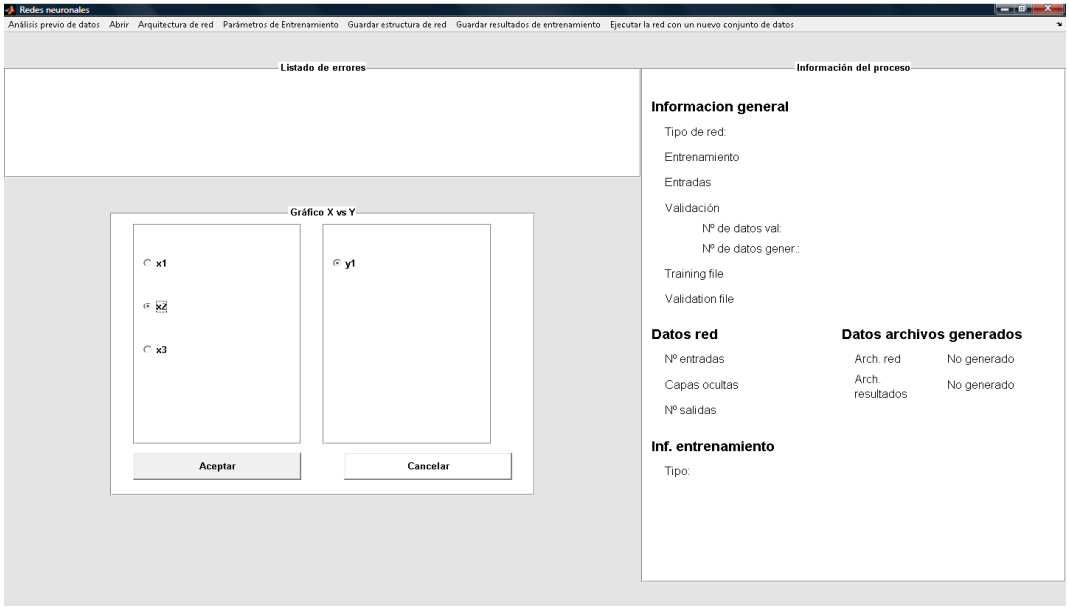

Figura 5.11: Panel para seleccionar las variables del gráfico X vs Y.

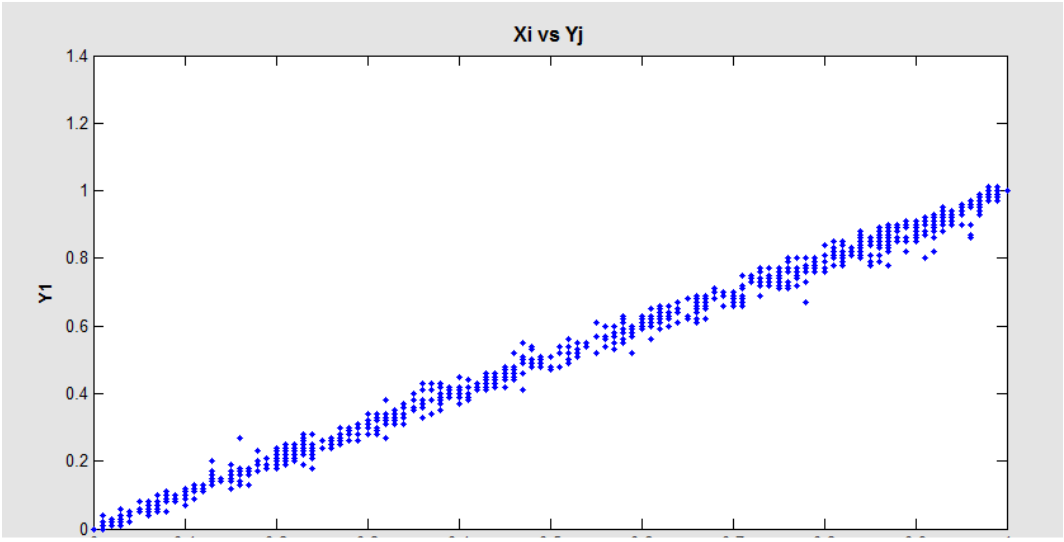

Figura 5.12: Gráfico de dispersión X vs Y.

representado en la gráfica 5.12, el usuario puede inferir que la salida depende linealmente de la entrada elegida. Además, podría ayudar al usuario a descubrir la existencia de posibles

datos erróneos en el fichero, si éste visualiza puntos muy alejados de la tendencia mostrada por la mayoría, es decir no se encuentran donde se concentran la mayoría de los datos.

***Boxplots*:** Esta opción permite visualizar los gráficos boxplot de las variables de entrada y salida. En dicho gráfico se representa una caja por cada variable que se grafique. Dicha caja está dibujada de forma que la posición de la línea roja respecto del eje Y coincida con la media de la muestra que corresponde a dicha variable, y las líneas azules inferiores y superiores correspondan al percentil 25 y 75 respectivamente. Los extremos inferior y superior corresponden al mínimo y máximo de la muestra. Un ejemplo de este tipo de gráficos se muestra en la figura 5.13 para las entradas. Es análogo para las salidas.

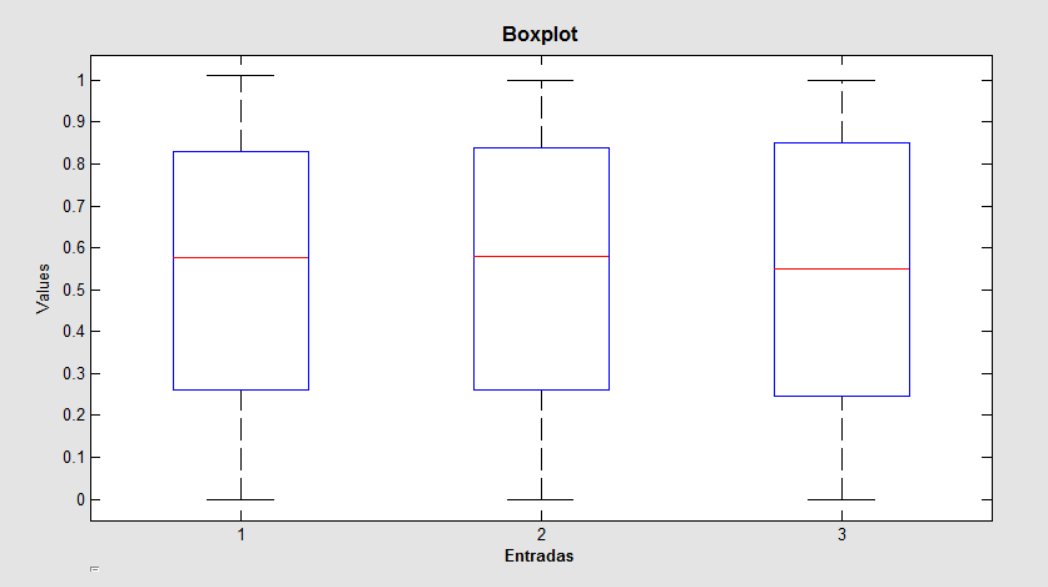

Figura 5.13: Gráfico Boxplot

***Modelo de regresión lineal múltiple*:** Esta opción permite estimar un modelo de regresión lineal múltiple. También puede ser útil para ver posibles datos atípicos, realizando en este caso un análisis multivariante, y no univariante como el realizado hasta ahora con las opciones previas. Este menú contiene las siguientes herramientas: *Estimar modelo*, *Histograma de los residuos*, *Gráfico Y real- Y estimada*, *Gráfico Yestimada vs residuos e Identificación de residuos con valores altos*.

Estimar modelo: Esta herramienta permite al usuario obtener las ecuaciones que relacionan las entradas con las salidas obtenidas de aplicar un modelo de regresión lineal múltiple. Cuando se ejecuta dicha herramienta, el programa mostrará un panel similar al que se muestra en la figura 5.14.

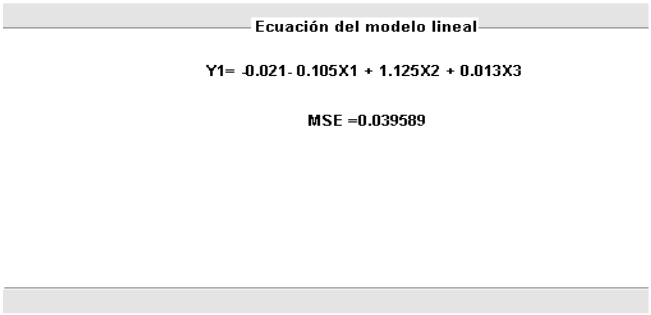

Figura 5.14: Panel con parámetros del modelo de regresión lineal múltiple

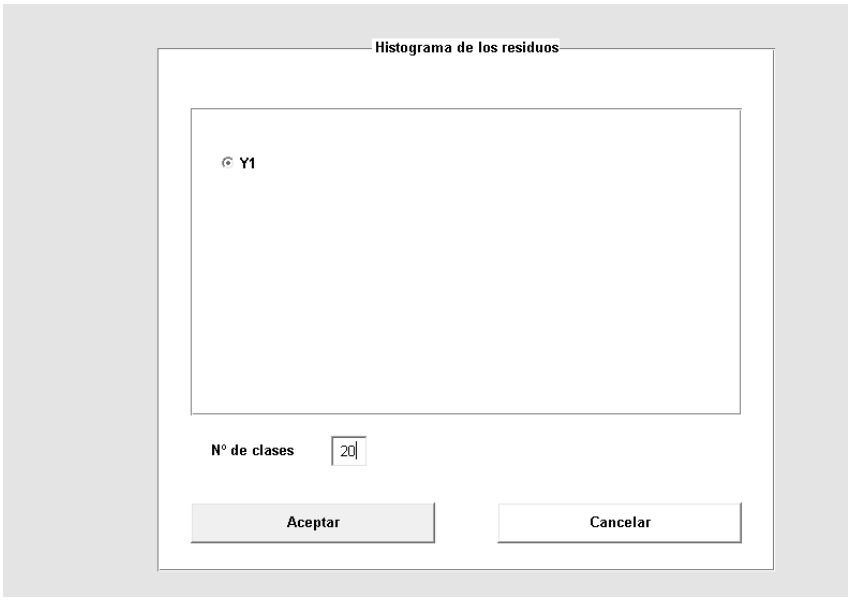

Figura 5.15: Panel para realizar el histograma de los residuos

Histograma de los residuos: Esta herramienta permite al usuario visualizar un histograma de los residuos del modelo de regresión lineal múltiple. El análisis de los residuos permite ver la bondad del ajuste estimado, si es bueno o no. Cuando seleccionamos esta herramienta se despliega un panel como el de la figura 5.15. En él se solicita que se especifique la variable salida para la cual queremos obtener los residuos, y el número de clases que queremos representar en el histograma. Una vez pulsado el botón aceptar aparece un gráfico similar al mostrado en la figura 5.16.

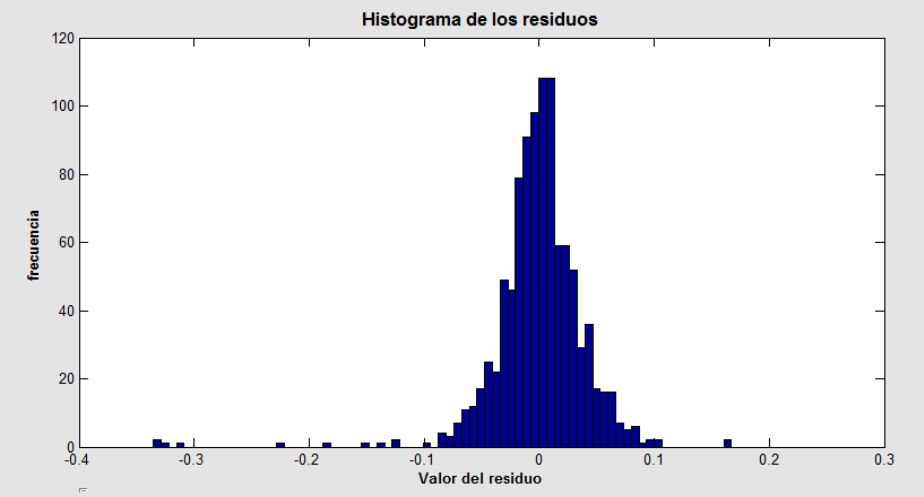

Figura 5.16: Histograma de residuos

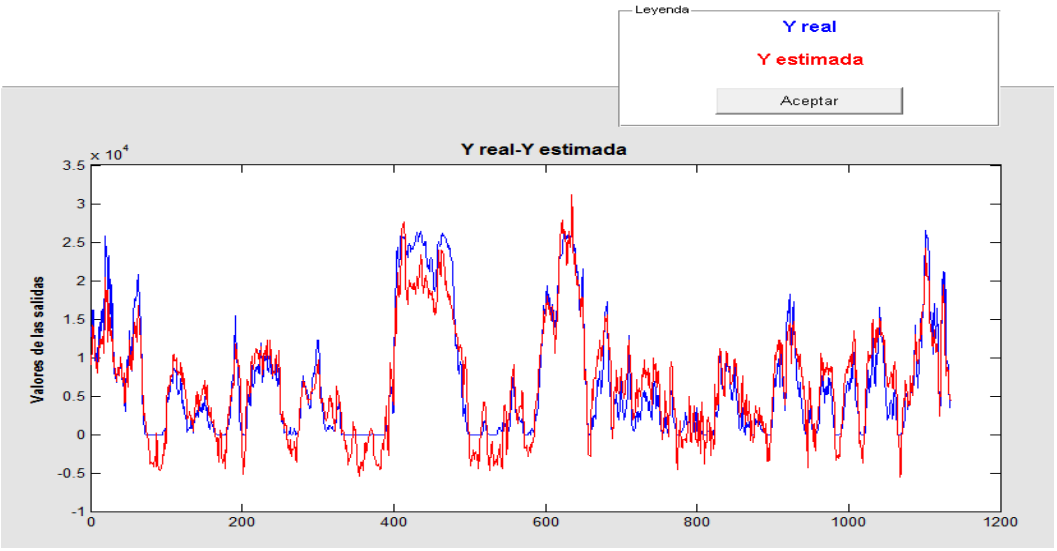

Figura 5.17: Gráfico Y real - Y estimada.

Gráfico Yreal -Yestimada: Esta herramienta muestra un gráfico que representa los valores reales frente a los valores estimados por el modelo lineal. Cuando se selecciona esta herramienta inicialmente aparece un panel similar al de la figura 5.15, donde únicamente hay que especificar la variable de salida que se quiere analizar. Inmediatamente después de pulsar el botón *Aceptar* aparecerá un gráfico similar al de la figura 5.17.

Gráfico Yestimada vs residuos: Esta herramienta permite visualizar un gráfica de dispersión entre los valores de salida estimados con el modelo de regresión lineal frente a los residuos de dicho modelo. Este gráfico permite ver si el ajuste lineal es bueno o no, si existe no linealidad, si hay heterocedasticidad y también es posible visualizar residuos con valores absolutos muy altos que podrían ayudar a identificar valores atípicos, aún cuando el modelo

lineal no fuera el más adecuado. Cuando se accede a esta opción, inicialmente se despliega un panel similar al de la figura 5.15, donde se tiene que especificar la salida de la cual queremos obtener este gráfico. Una vez realizado esto, el programa mostrará un gráfico similar al de la figura 5.18.

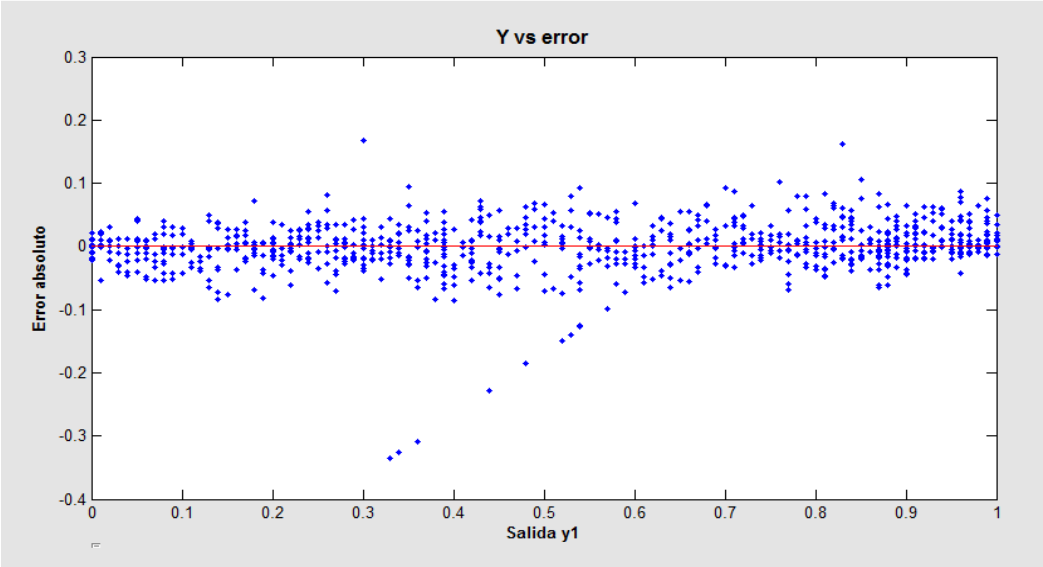

Figura 5.18: Gráfico de dispersión Yestimada vs residuos

Identificación de observaciones con residuos muy altos: Esta opción permite identificar qué observaciones (qué datos) producen un residuo muy alto, en valor absoluto, con respecto al total de residuos del modelo lineal. Estas observaciones podrían ser observaciones atípicas.

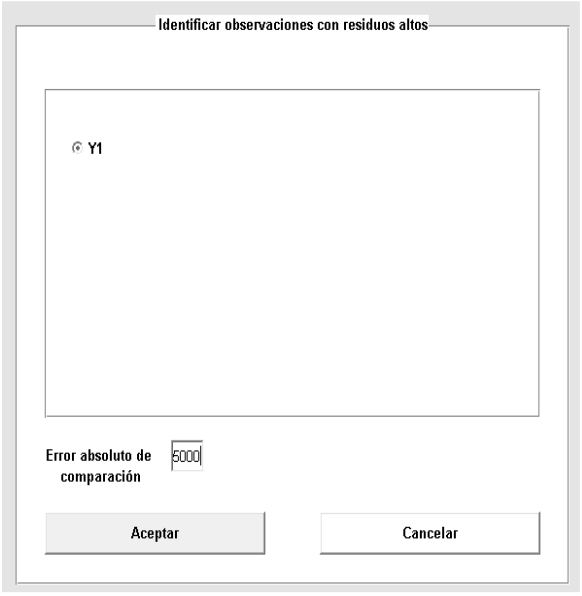

Figura 5.19: Panel para la identificación de residuos altos.

Cuando se ejecuta esta herramienta, el programa muestra un panel similar al mostrado en la figura 5.19. En él tenemos que seleccionar qué variable de salida se analizará y qué valor de los residuos se considera como máximo. Las observaciones con residuos mayores, en valor absoluto, serán identificadas mediante esta opción. Una vez elegida, el programa generará un archivo que se encontrará en una carpeta denominada *Análisis previo*. Dicha carpeta se encontrará en el directorio de trabajo. Este archivo contendrá las observaciones con residuos altos, así como el número de observación (fila) del fichero analizado.

5.3.4.-Arquitectura de red.

En este grupo de funciones se encuentran las herramientas necesarias para definir la arquitectura de los dos tipos de redes con las que se trabaja en el programa, redes backpropagation y redes de base radial (figura 5.20).

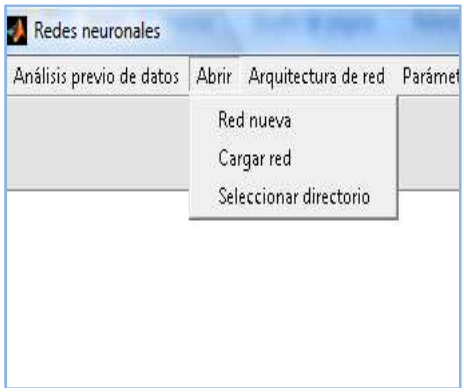

Figura 5.20: Menú Arquitectura de red

***Backpropagation*:** Cuando se selecciona dicha opción se abre un panel para definir la arquitectura de la red. En dicho panel (figura 5.21), hay que definir primero el número de capas ocultas que tendrá dicha red.

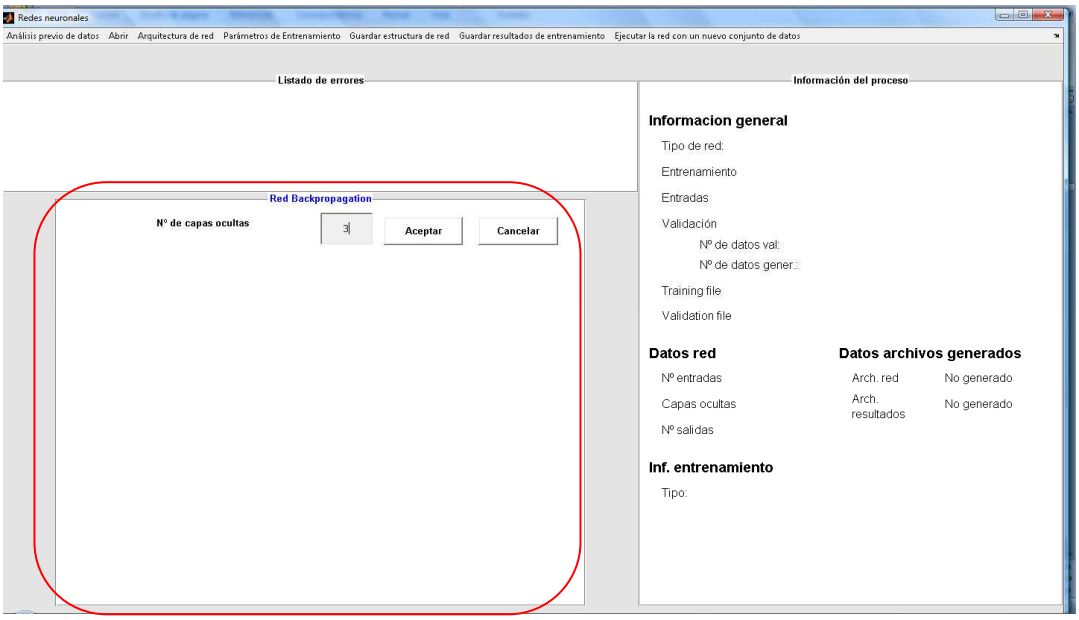

Figura 5.21: Panel I para definir la arquitectura de la red Backpropagation

Una vez indicado el número de capas ocultas se pulsa el botón *aceptar* y se despliega un conjunto de casillas para introducir los datos de cada una de las capas, incluidas la de entrada y salida de la red. Ver figura 5.22. Los parámetros que se deben definir son: el número de neuronas de entrada, el número de neuronas de cada capa oculta y sus respectivas funciones de activación, y el número de neuronas de salida y su función de activación. Una vez introducido los parámetros deseados, se pulsa el botón *aceptar* de la parte inferior.

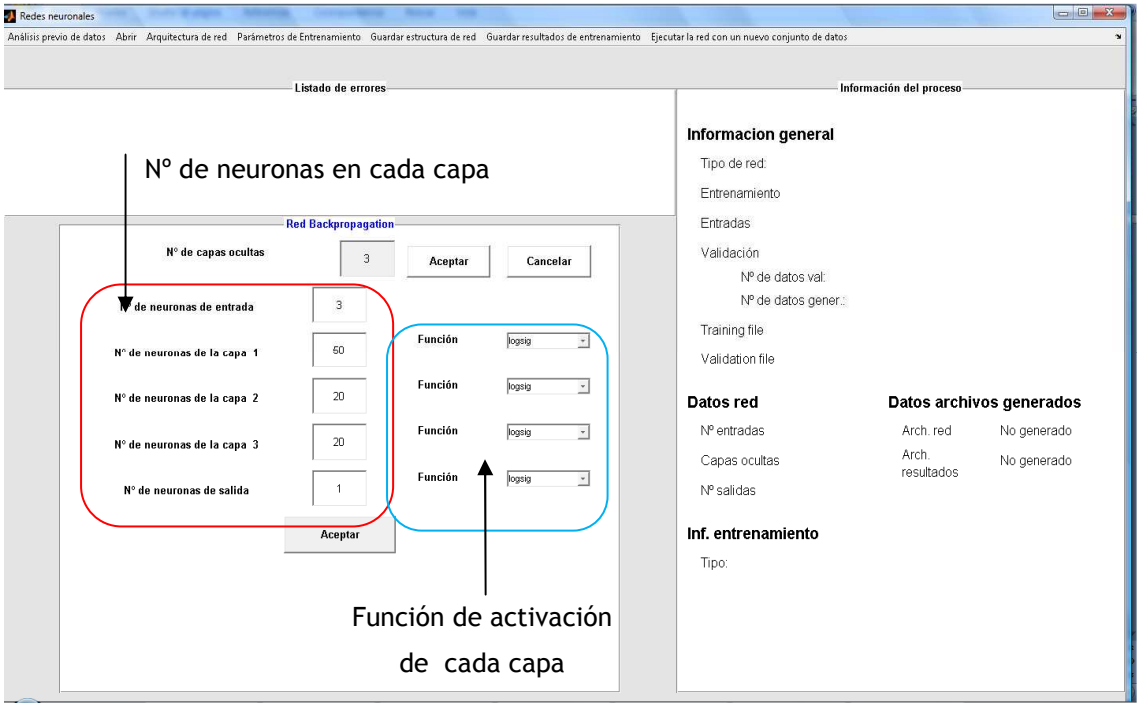

Figura 5.22: Panel II para definir la arquitectura de la red Backpropagation

***Base radial*:** Con esta herramienta se define la arquitectura de la red de base radial. Cuando se selecciona se despliega el panel que se muestra en la figura 5.23. En él se pide que se introduzca el número de neuronas de la capa entrada y de salida. Asimismo se debe introducir el máximo número de neuronas ocultas que puede tener la red (para ver para qué se utiliza este parámetro mirar el apartado 3.2.2.) y el valor mínimo de error que se quiere obtener al entrenar la red. Si se alcanza este error la red se deja de entrenar, pero podría darse el caso de que no se alcance este valor, lo cual será indicado por el programa.

Además de estas opciones, se debe definir aquí el parámetro *spread* (mirar apartado 3.2). Hay dos opciones: la primera es seleccionar un solo valor para dicho parámetro, *Spread único*; la segunda seleccionar un intervalo y un incremento para realizar el entrenamiento con varios valores de spread, *Spread intervalo*. Esta permite visualizar de una sola vez el comportamiento de la red de base radial ante varios spreads, lo cual facilita decidir cuál es el valor más óptimo para éste parámetro.

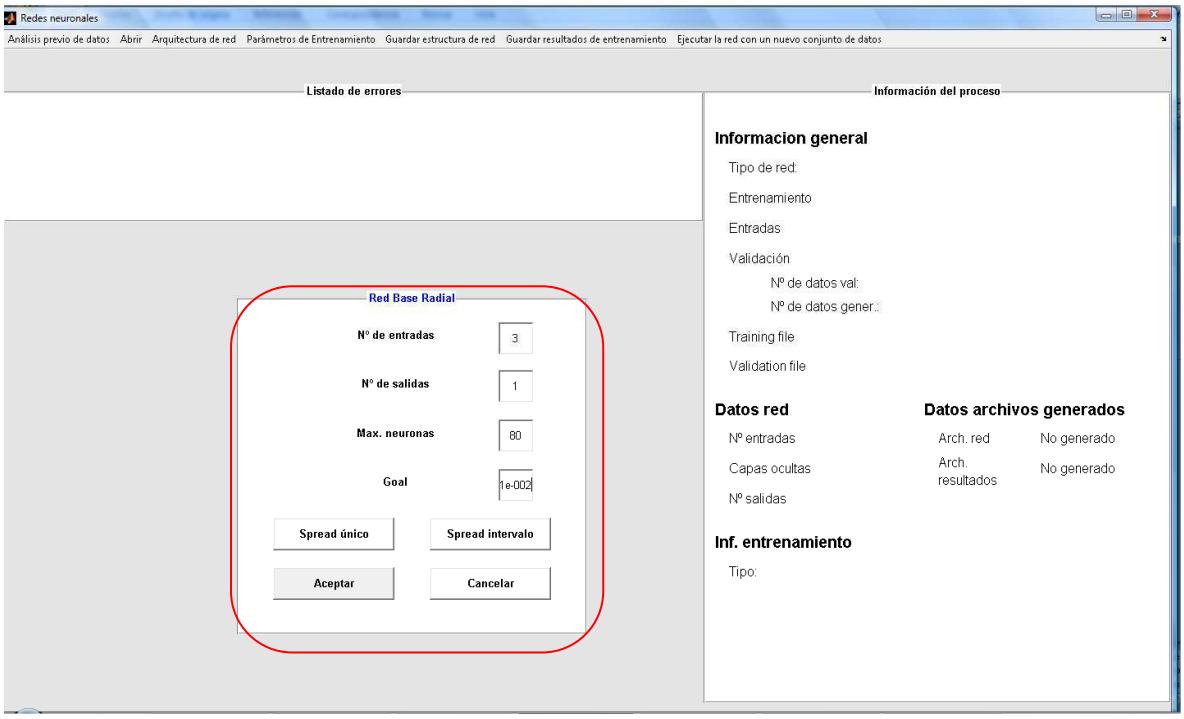

Figura 5.23: Panel para definir la arquitectura de la red de base radial

Cuando se pulsa uno de los dos botones para definir el spread se despliega una ventana que dependerá del botón que se pulse. Si se pulsa el botón *Spread único* se despliega la ventana que se visualiza en la figura 5.24. En ella únicamente se debe indicar el valor del spread. Una vez introducido se debe pulsar el botón *aceptar*. Si se pulsa el botón *Spread intervalo* se despliega la ventana que aparece en la figura 5.25. En ella hay que definir el intervalo de valores para el spread que se quiere estudiar; para ello se define el *spread inicial* y el *spread final*, y el paso que existirá entre los distintos valores de spread; para ello hay que definir el

parámetro *Salto del spread*. Como ocurría con el caso anterior, una vez fijado todos los parámetros se debe pulsar el botón *aceptar*. Por último, y una vez fijado todos los parámetros que definen la arquitectura de una red de base radial, se pulsa el botón *aceptar* del panel inicial.

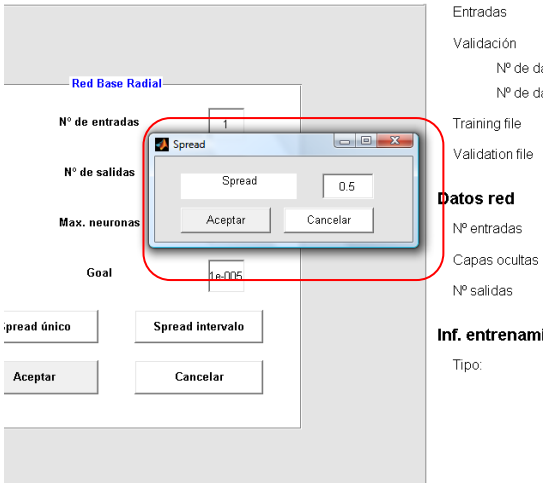

Figura 5.24: Ventana para seleccionar un único spread

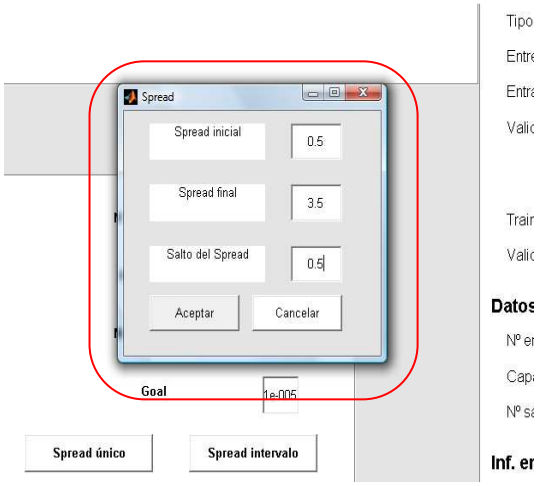

Figura 5.25: Ventana para seleccionar el intervalo

Es importante tener en cuenta que el Matlab utiliza el algoritmo OLS para optimizar la red de base radial. Este algoritmo permite optimizar de forma automática el número de neuronas de la capa oculta y los centros de dichas neuronas para un valor de *spread* específico. Por este motivo, el *spread* es el único parámetro que se fija en este tipo de redes. El algoritmo minimiza siempre el error cuadrático medio.

5.3.5.-Parámetros de entrenamiento.

Esta opción permite definir completamente la fase de entrenamiento y además permite ejecutar dicha fase. Contiene las siguientes herramientas: *error*, *algoritmo de optimización*, *tratamiento de entradas*, *cross validation*, *cargar patrones* y *entrenar*, tal y como se ve en la figura 5.26. Cabe destacar que las dos primeras opciones sólo están disponibles cuando se trabaje con redes backpropagation. En el caso de redes de base radial, el algoritmo de optimización es siempre el algoritmo OLS y el error que se minimiza es el error cuadrático medio.

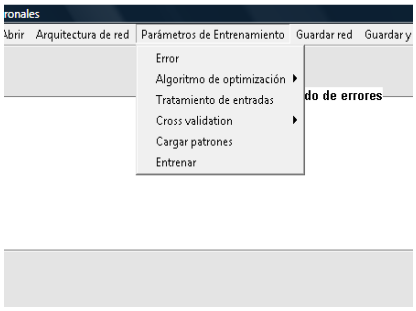

Figura 5.26: Menú Parámetros de Entrenamiento

Error: Esta herramienta permite seleccionar al programa el tipo de error que se va a minimizar durante la fase de entrenamiento de la red backpropagation .

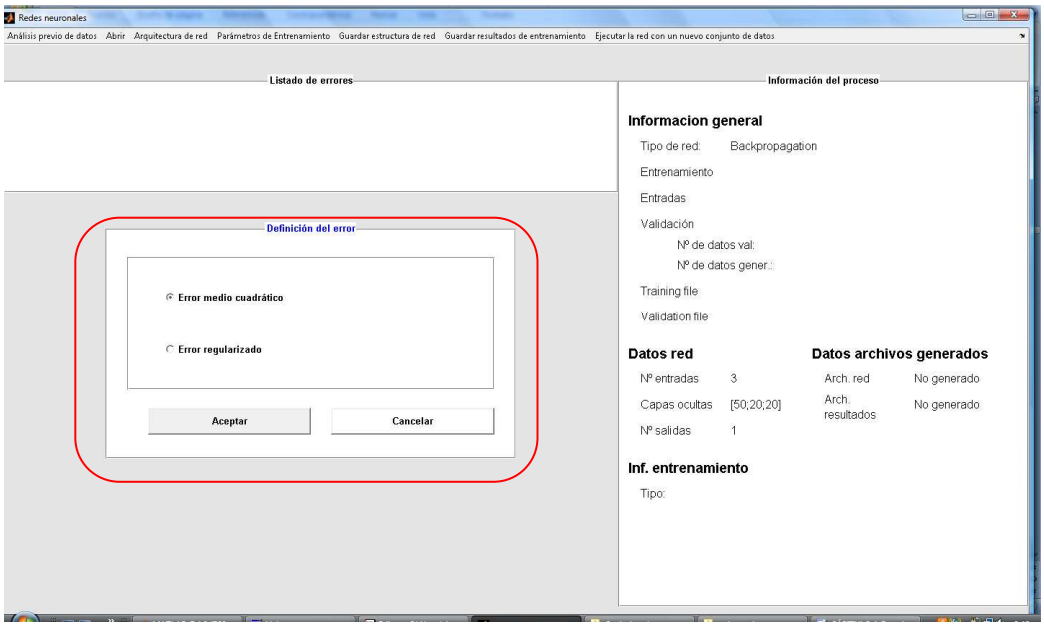

Figura 5.27: Panel para seleccionar el error

Cuando se accede a esta herramienta se despliega un nuevo panel, tal y como se indica en la figura 5.27. En dicho panel se presentan dos opciones. La primera es *Error cuadrático medio* (MSE) que corresponde con la definición del error dada en el apartado 3.1.3:

$$E(n) = \frac{1}{N}\sum_{k=1}^{N}\sum_{n=1}^{r}(s_k(n) - y_k(n))^2$$

La otra es *Error regularizado*, que se corresponde con la definición dad en el apartado 3.1.4:

$$msereg = \gamma \cdot mse + (1-\gamma)msw$$

$$msw = \frac{1}{K}\sum_{j=1}^{K} {w_j}^2$$

donde γ es un parámetro que se debe introducir si se selecciona este error, y mse la función error definida anteriormente.

No es obligatorio seleccionar un tipo de error. Por defecto el error será el error cuadrático.

***Algoritmo de optimización*:** Como se comentó al principio de este apartado, esta opción únicamente está disponible cuando se trabaja con redes backpropagation. Esta herramienta permite elegir el algoritmo de optimización que se va a utilizar durante el entrenamiento de la red, así como la definición de los parámetros de dicho algoritmo. Para más información sobre los tipos de entrenamiento mirar los apartados 2.3.3 y 2.3.4.

Los diferentes algoritmos de optimización que se pueden utilizar en esta aplicación se han agrupado según sus características. La clasificación que se ha realizado es la siguiente:

Tabla 5.1: Clasificación de los métodos de entrenamiento utilizados

Algoritmos básicos	**Mejoras del algoritmo de la regla delta**	**Algoritmos del gradiente conjugado**	**Algoritmos de cuasi Newton**	**Algoritmo de Levenberg**
Regla delta generalizada	Propagación elástica	Fletcher-Reeves	BFGS	Método de Levenberg
Regla delta generalizada con momento	Ratio de aprendizaje variable	Polak-Ribière	Algoritmo de la secante.	
		Powell-Beale		
		Gradiente conjugado escalado		

Estos métodos de entrenamiento están disponibles mediante menús desplegables a partir del menú *Algoritmo de optimización* según esta clasificación.

Cada entrenamiento tiene asociado unos determinados parámetros, que en muchos casos coinciden entre sí. Una vez seleccionado un algoritmo se desplegará un panel donde se

debe introducir el valor de todos sus parámetros. Por defecto el programa asigna unos valores que resultan adecuados para la mayoría de los casos, por lo que si se desconoce qué valor inicial asignar a un determinado parámetro se pueden dejar el valor por defecto, y posteriormente, si se desea modificarlo, para comparar el comportamiento de la red.

A continuación se describirá los parámetros que se pueden definir para cada uno de los algoritmos y su significado.

- Regla delta generalizada.

Tabla 5.2: Resumen de parámetros del algoritmo de regla delta

Parámetro	Significado
Ratio de aprendizaje	Es el valor del parámetro α (mirar apartado 2.3.3).
GOAL	Si durante el entrenamiento el error cometido por la red es inferior a este valor en alguna época, se detiene el entrenamiento.
Épocas	Máximo número de veces que se presentan los datos de entrenamiento a la red.
Max_fail	Máximo número de épocas en las cuales el error de validación puede crecer antes de que se detenga el entrenamiento.

- Regla delta generalizada con momento.

Tabla 5.3: Resumen de los parámetros del algoritmo de la regla delta con momentum

Parámetros	Significado
Ratio de aprendizaje	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
GOAL	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
Épocas	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
Max_fail	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
Momento	Es el valor del parámetro η (mirar apartado 2.3.4)

- Propagación elástica.

Tabla 5.4: Resumen de los parámetros del algoritmo de propagación elástica

Parámetros	Significado
Delta 0	Es el valor en la primera iteración de Δ_{ij} (mirar ecuación 2.23)
Amplificación	Es el factor de amplificación que se aplica a Δ_{ij} si se cumplen las condiciones de la expresión 2.23
Reducción	Es el factor de minoración que se aplica si se cumplen las condiciones de la expresión 2.23
Delta max	El valor máximo que puede alcanzar Δ_{ij}
GOAL	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
Épocas	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
Max_fail	Tiene el mismo significado que en el algoritmo de la regla delta generalizada.

- Ratio de aprendizaje variable.

Tabla 5.5: Resumen de los parámetros del algoritmo del ratio de aprendizaje variable

Parámetros	Significado de los parámetros
Ratio de aprendizaje	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
GOAL	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
Épocas	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
Max_fail	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
Amplificación de alpha	Es el factor de amplificación que se aplica al ratio de aprendizaje según el apartado 2.3.4.
Reducción de alpha	Es el factor de reducción que se aplica al ratio de aprendizaje según el apartado 2.3.4.

Momento	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).

- Algoritmos del gradiente conjugado y de cuasi-Newton.

Todos los algoritmos de estos dos grupos tienen los mismos parámetros con el mismo significado, que ya se ha visto anteriormente con otros métodos.

Tabla 5.6: Resumen de los parámetros de los algoritmos del gradiente conjugado y de cuasi-Newton

Parámetros	Significado
GOAL	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
Épocas	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
Max_fail	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).

-Algoritmo de Levenberg.

Tabla 5.7: Resumen de los parámetros del algoritmo de Levenberg.

Parámetros	Significado
Ratio de aprendizaje	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
GOAL	Tiene el mismo significado que en el algoritmo de la regla delta generalizada (tabla 5.2).
Épocas	Tiene el mismo significado que en el algoritmo de la regla delta generalizada

	(tabla 5.2).
Max_fail	Tiene el mismo significado que en el algoritmo de la regla delta generalizada.
Amplificación de mu	Factor ß por el que se multiplica μ según se describe en el apartado 2.3.4.
Reducción de mu	Factor ß de reducción que se aplica a μ según se describe en el apartado 2.3.4.
Máximo mu	Máximo valor que puede alcanzar μ

En la GUI desarrollada se han introducido todos los algoritmos de optimización disponibles en Matlab para las redes Backpropagation. Se ha intentado proporcionar al usuario facilidad de uso de dichos métodos proporcionando valores por defecto razonables, pero mostrando al usuario que existen, y a la vez flexibilidad porque son fáciles de cambiar y ver cómo se comporta la red ante ese cambio.

Tratamiento de entradas: Esta herramienta permite seleccionar cómo se quiere que la red trate los valores de las variables de entrada y salida. Es decir, permite seleccionar si la red va a entrenar utilizando el valor real de las entradas, o si se les va aplicar alguna transformación para mejorar el entrenamiento (mirar apartado 3.1.5.).

El programa da la opción de aplicar dos transformaciones y de no aplicar ninguna transformación. Las transformaciones que se pueden aplicar son:

Escalada: Se escalan las entradas de forma que pertenezcan en el rango [-1,1] o [0,1].

Estandarizada: Los valores se estandarizan de manera que tengan media cero y desviación típica 1.

Por defecto el programa aplica la misma transformación a las variables salidas que la que se ha aplicado a las variables de entrada de la red. Por otro lado, cabe destacar que el usuario no se tiene que preocupar si las salidas que muestre la red están transformadas o no. Por defecto, todas las salidas que pueda dar la red se les ha aplicado ya una transformación inversa. Únicamente el *mse* conserva la transformación para una mejor visualización.

Para seleccionar el tipo de transformación que se desea, únicamente hay que acceder a la herramienta *Tratamiento de entradas*. Entonces, se desplegará un panel tal y como se muestra en la figura 5.28. En él aparecen tres opciones para elegir una, *Sin transformar*,

Escalada y *Estandarizada*. Se elige una y se pulsa el botón *Aceptar*. Inmediatamente después se verá la opción escogida en el panel *Información del proceso*.

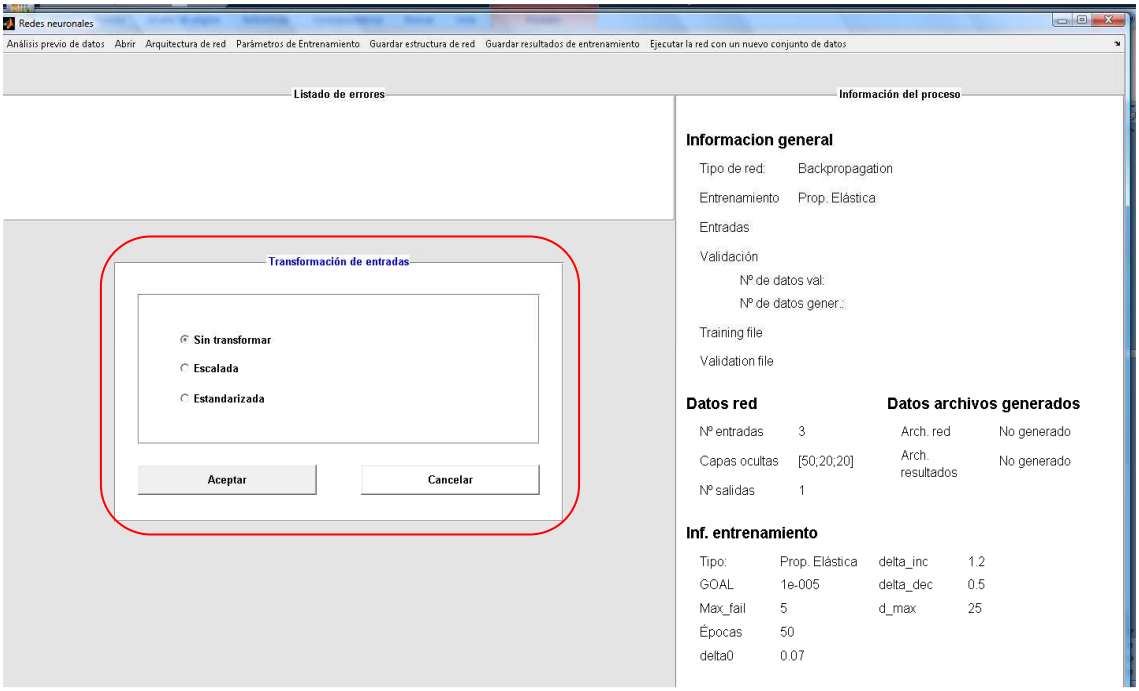

Figura 5.28: Panel para seleccionar el tipo de transformación para las entradas.

Cross Validation: Esta herramienta permite definir si se va a realizar sólo entrenamiento de la red o si también se realizará una fase de generalización. De ser así, permite seleccionar cómo se va a realizar la fase de generalización. Antes de explicar las opciones que se tienen, es necesario explicar los diferentes tipos de datos que distingue la GUI:

Datos de entrenamiento: Son los datos que utiliza el programa para el entrenamiento de las redes. Por defecto son todos los datos del fichero, en el programa no se especifican directamente.

Datos de validación: Sólo se usan en el caso de las redes backpropagation, durante la fase de entrenamiento, para mejorar el algoritmo. Durante esta fase, se calcula el error cometido en los datos de validación, de manera que el algoritmo se detiene en el caso de que éste se ha incrementado un número de veces consecutivas (para más información mirar el apartado *Parada temprana* en 3.1.4.).

Datos de generalización: Estos datos se utilizan una vez que la red ha finalizado la etapa de entrenamiento. Son datos nuevos para la red y permiten comprobar el grado de generalización de la misma, es decir, si la red es capaz de predecir valores de salida buenos para valores de entrada no conocidos hasta el momento. Permite comprobar si la red es capaz

de generalizar lo que ha aprendido. Este tipo de datos se pueden utilizar para ambos tipos de redes.

Una vez explicado esto ya se puede explicar las 3 opciones de *Cross validatio*n:

Entrenamiento simple, Sólo se realiza la fase de entrenamiento. En el caso de redes backpropagation este entrenamiento consiste en seleccionar del total de datos contenidos en un fichero, un conjunto de datos de validación a usar durante el entrenamiento. La selección de estos datos es aleatoria. El programa da la opción de indicar cuantos datos de validación se van a usar mediante un panel, como el de la figura 5.29, el resto serán datos de entrenamiento. En el caso de redes de base radial, todos los datos del fichero se usan como patrones de entrenamiento. No se realiza fase de generalización. El fichero con los datos se debe cargar desde la opción *Cargar patrones*, como se verá a continuación.

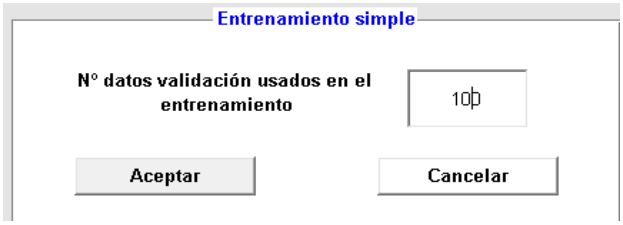

Figura 5.29: Panel para indicar datos de validación en el entrenamiento simple y generalización externa

Entrenamiento simple con generalización externa: Esta opción complementa la opción de entrenamiento simple. Consiste en realizar el entrenamiento simple y comprobar la capacidad de generalización de la red con un conjunto de datos distintos, contenido en otro fichero. Este fichero deberá ser cargado también desde *Cargar patrones*. En el caso de redes backpropagation, una vez seleccionada la opción *Generalización externa*, aparecerá nuevamente el panel de la figura 5.29, para indicar los datos de validación que serán usados durante el entrenamiento simple. En el caso de redes de base radial, igual que antes, no aparecerá ningún panel para el entrenamiento simple.

Cross validation: La tercera opción y más importante es realizar lo que se conoce como *cross validation*. Mediante esta opción se realiza varias veces el entrenamiento y la generalización de la red. Para ello se usan los datos de un solo fichero. El total de estos datos se divide en dos conjuntos: uno de ellos es utilizado en la fase de entrenamiento y el otro es usado para la fase de generalización de la red. La división se hace en el caso de ésta GUI, de forma aleatoria.

La división de datos en patrones de entrenamiento y generalización es muy importante en la modelización, ya que dependiendo de cómo sean estos datos se obtendrán resultados distintos, sobre todo si el número total de datos no es muy alto. Por este motivo se recomienda realizar el entrenamiento de la red con distintos conjuntos de entrenamiento y validación. Es decir, hacer cross-validation un número suficiente de veces. De esta manera se obtiene una mejor visión del comportamiento de la red que no depende de la "suerte" con la que han sido elegidos los datos de entrenamiento y los de generalización.

Al seleccionar esta opción aparecerá el panel mostrado en la figura 5.30, donde es necesario especificar el número de datos que van a ser usados en la fase de generalización, y en el caso de redes backpropagation, el número de datos de validación. También se debe indicar el número de veces (iteraciones) que se realizará cross-validation.

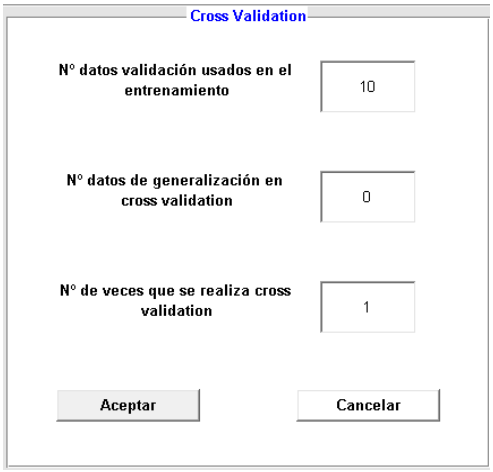

Figura 5.30: Panel para especificar el número de datos para cross-validation

Una vez que se haya elegido cualquiera de éstas tres opciones hay que pulsar el botón *Aceptar*. Inmediatamente después aparecerá la información pertinente en el panel *Información del proceso*.

Cargar patrones: Con esta herramienta se indica el nombre de los ficheros que se van a utilizar en el entrenamiento y generalización de la red. Es importante volver a señalar que el usuario debe asegurarse de que estos ficheros se encuentren en la carpeta seleccionada como directorio al principio de la creación de la red. Por otro lado, estos ficheros deben ser un documento de texto para que puedan ser leídos por el programa. Además también deben poseer una estructura adecuada. Dicha estructura consiste en agrupar los datos en filas.

Cuando se selecciona esta opción se despliega un panel en la ventana del programa tal y como se muestra en la figura 5.31. En ella aparecen dos campos para especificar el nombre de dos archivos (no se debe introducir la extensión del archivo). En el campo superior se señala el fichero que contiene los datos que serán usados en las opciones *Entrenamiento Simple* y *Crossvalidation*. En el campo inferior se señala el archivo que contiene los datos de generalización que serán usados en el caso de usar la opción *Generalización externa*. Este fichero sólo se debe especificar en el caso de que se utilice dicha opción.

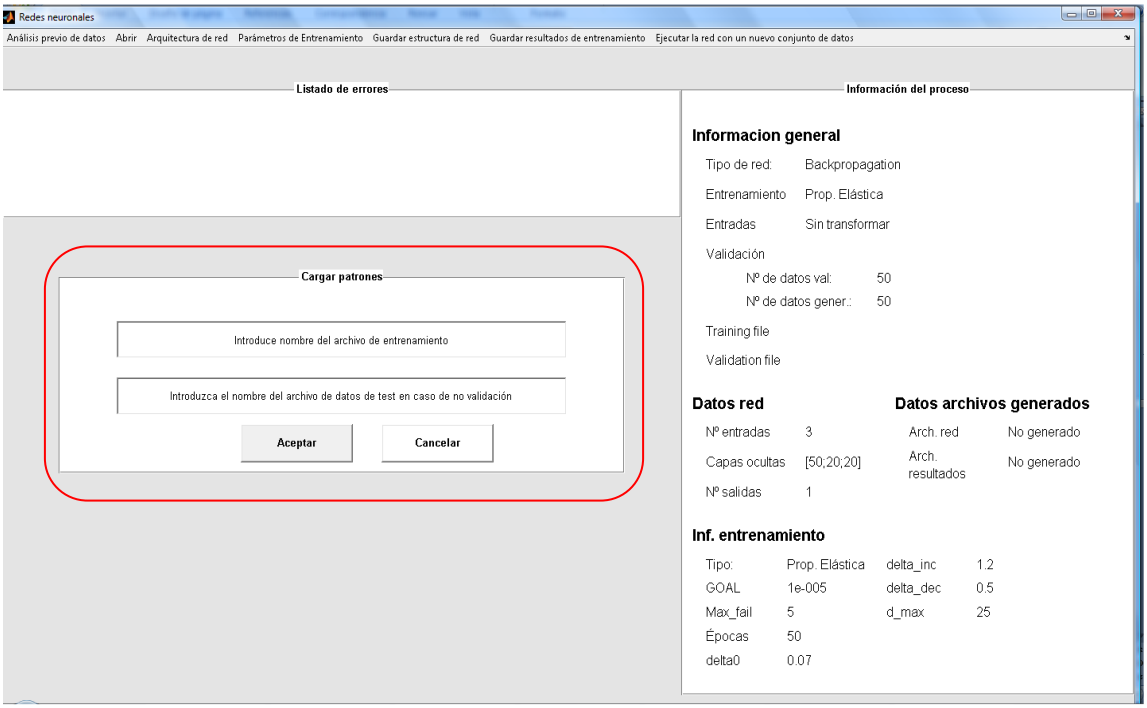

Figura 5.31: Panel para la especificación de ficheros de datos.

Una vez introducido el nombre del archivo, se pulsa el botón *Aceptar*. Seguidamente aparecerá un panel similar al de la figura 5.7b donde se especificarán las columnas que son entradas y las columnas que son salidas. Posteriormente se verá los ficheros seleccionados en el panel *Información del proceso*.

Entrenar: Una vez definido la red y el entrenamiento con todo lo que se ha visto en este apartado, y si todo se ha realizado correctamente, el programa está listo para entrenar la red. Mediante esta opción se inicia el entrenamiento y generalización de la red según se haya definido previamente. Según el tipo de red y el tipo de entrenamiento que se esté realizando se visualizará en la pantalla unas determinadas gráficas que a continuación se describirán. Así tendremos los siguientes casos:

Redes Backpropagation o Redes de Base radial con un solo valor de *spread*:

- Si se realiza *Entrenamiento simple* el programa no visualizará nada durante el proceso. Cuando haya terminado mostrará una ventana que indica *Entrenamiento finalizado* y el error de entrenamiento, tal y como se muestra en la figura 5.32.

Figura 5.32: Ventana que indica que el entrenamiento ha finalizado

- Si se realiza *Entrenamiento simple y generalización externa*, se visualizará lo mismo que en el caso anterior más el error de generalización cometido por la red ante los datos del fichero de generalización.

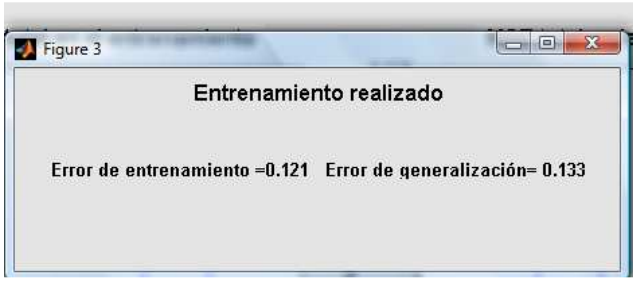

Figura 5.33: Ventana que indica que el entrenamiento ha finalizado

- Si se realiza *Cross validation*, se visualizarán las gráficas que se muestran en la figura 5.34. La gráfica de la izquierda muestra el error cuadrático medio total de la red en la fase de entrenamiento (tiene en cuenta el error para todas las variables salidas) y por cada iteración. La gráfica de la derecha representa el error cuadrático medio total en la fase de generalización y por cada iteración. Por otro lado al final del entrenamiento se vuelve a mostrar la ventana que aparece en la figura 5.33, donde los valores se han calculado como la media de todas las iteraciones.

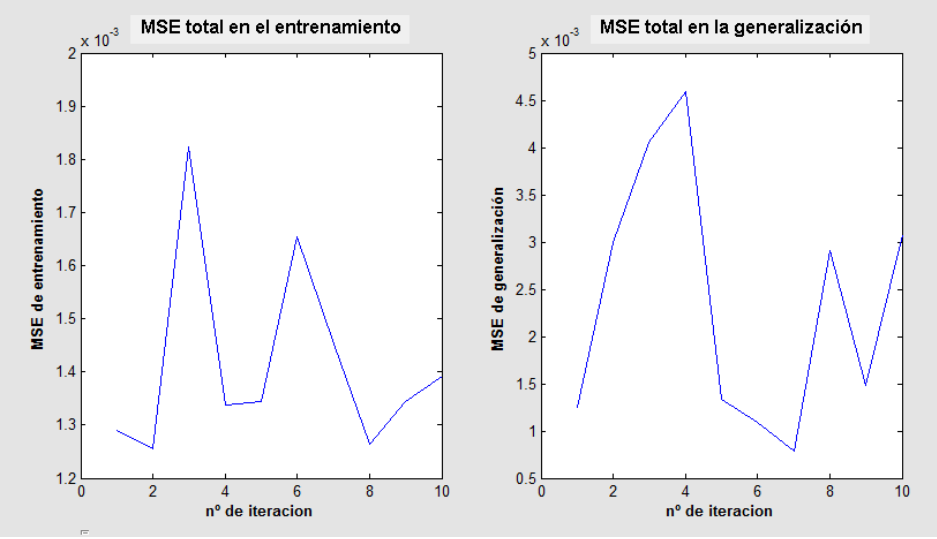

Figura 5.34: Gráficas que se muestran durante el entrenamiento Cross validation

Redes de base radial con varios valores de Spreads

- Dependiendo del entrenamiento se muestra dos gráficas (si existe generalización) relativas al error de entrenamiento y generalización, o únicamente la relativa al error de entrenamiento si se realiza entrenamiento simple. Dicho error se representa en función del valor del spread. El error medio se calcula teniendo en cuenta el error para todos los datos, todas las salidas, y en el caso de que se trabaje con cross validation se tienen en cuenta también todas las iteraciones que se realicen en este entrenamiento. Al final vuelve a aparecer la ventana de la figura 5.32 ó 5.33. Los valores esta vez son la media de todas las iteraciones realizadas.

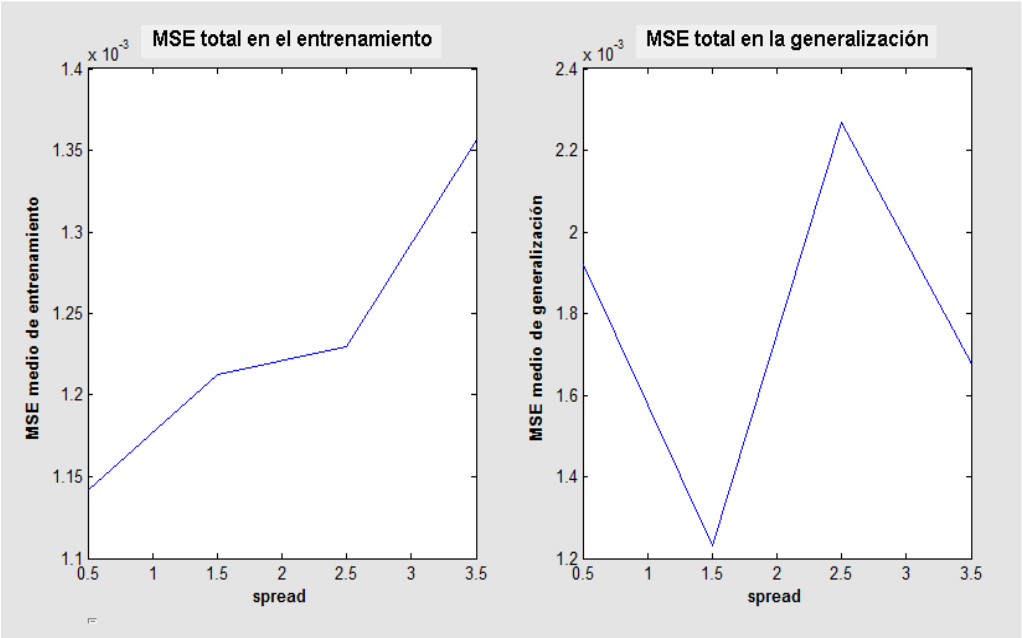

Figura 5.35: Gráficas que se muestran durante el entrenamiento de las redes de base radial cuando se trabaja con spread variable

Mediante el gráfico de la figura 5.35 se intenta facilitar al usuario la selección del valor óptimo del Spreads.

Como se ven en las figuras 5.34 y 5.35 también existe la posibilidad de detener el entrenamiento en el caso de que los errores cometidos por la red sean excesivos. Este botón (llamado *stop training*) únicamente aparece cuando se ejecuta la funcionalidad de cross validation o se trabaja con redes de base radial con distintos valores de Spreads.

5.3.6.-Postproceso

Este menú está disponible cuando se haya realizado el entrenamiento y generalización de una red. Permite visualizar ciertos resultados que se han considerado importantes en este proyecto. En este apartado se explicará el significado de cada gráfica así como su utilidad a la hora de conocer el comportamiento de la red, y si este es apropiado para los datos que disponemos.

En este menú únicamente existe una opción denominada *Gráficas*, tal y como se ve en la figura 5.36. Cuando se selecciona, se despliega un panel denominado *Driver gráfica*. Dicho panel nos permite seleccionar el tipo de gráfica que deseamos y la salida de la red de la que queremos obtener los resultados. Dicho panel se muestra en la figura 5.37.

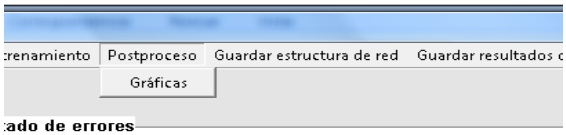

Figura 5.36: Menú de Postproceso

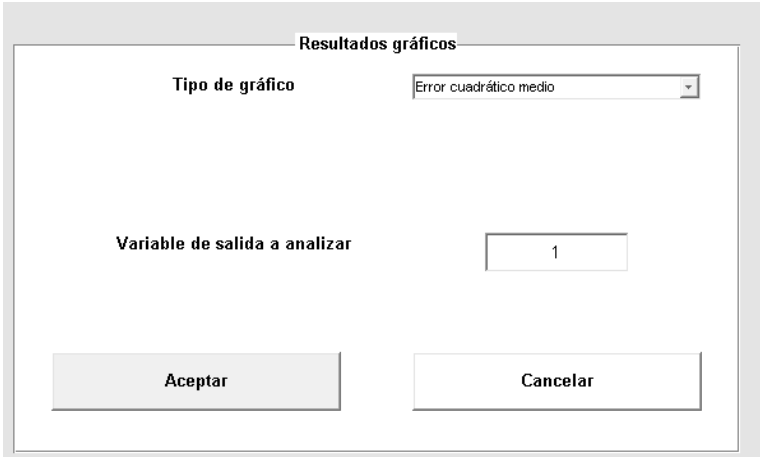

Figura 5.37: Panel *Driver gráfica*.

Los diferentes tipos de gráficas que se pueden visualizar son: *Error cuadrático medio (MSE), Error relativo absoluto medio (MAPE), Valores reales-Valores estimados en la fase de entrenamiento, Valores reales-Valores estimados en la generalización, Valores estimados vs residuos en fase de entrenamiento* y, *Valores estimados vs Residuos en generalización*. El tipo de gráfica se selecciona del menú desplegable en el campo *Tipo de gráfico* y la salida especificándola en el campo editable *Variable de salida a analizar*. A continuación se explica con detalle cada una de las gráficas.

Error cuadrático medio (MSE): Esta gráfica representa el error cuadrático medio que comete la red en las fases de entrenamiento y de generalización, en cada una de las iteraciones que se lleven a cabo en el entrenamiento cross validation. Por tanto, esta gráfica sólo mostrará algo en el caso de que utilicemos esta opción.

Es muy similar a la que muestra el programa durante el entrenamiento de las redes, ya que se representa el MSE para cada iteración, pero en este caso, el error es calculado para cada variable salida (antes era la media para todas las variables salidas). De esta manera el usuario puede estudiar más detalladamente el comportamiento de la red para cada variable salida. El MSE es el tipo de error que, en muchos casos, se utiliza para optimizar el comportamiento de la red.

Un ejemplo de este tipo de gráfica es el que se muestra en la figura 5.38. En ella se ven como existen dos curvas, una (la azul) asociada al error en la fase de entrenamiento, la otra (roja) asociada a los errores que comete la red en la generalización.

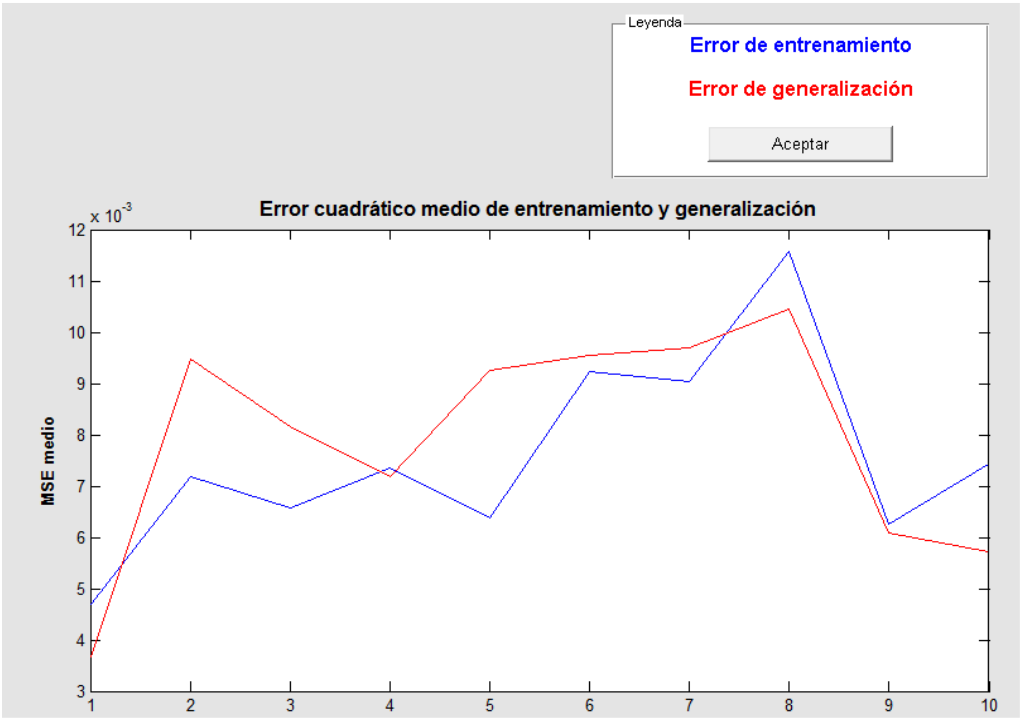

Figura 5.38: Gráfica del error de entrenamiento y test por iteración.

Esta gráfica sirve al usuario para verificar la aptitud de sus datos, es decir, le permite comprobar si el error que comete la red depende mucho de la división que se ha hecho de los datos, en conjunto de entrenamiento y conjunto de generalización. Si por ejemplo, ambos errores varían mucho (algún orden de magnitud) de una iteración a otra, quiere decir que la bondad del entrenamiento de la red depende mucho de la división de los datos. Sería adecuado quizás usar una muestra más grande y repetir muchas veces el entrenamiento con cross validation.

Por otro lado, también le permite ver al usuario las diferencias existentes entre el error de entrenamiento y el error de generalización. De esta manera, comprobar si la red generaliza bien los datos, o por el contrario ha sufrido lo que se conoce como sobreaprendizaje (ver apartado 3.1.4.), donde las redes son capaces de aproximar muy bien los datos que les han sido mostrados durante el entrenamiento, pero sin embargo, no es capaz de adaptarse a nuevos datos que se le presenten. Sin embargo, lo que sí es normal es que el error que se comete ante los datos de generalización sea mayor al error de entrenamiento.

Error relativo absoluto medio (MAPE): Esta gráfica es similar a la anterior. Sin embargo en este caso se representa el error relativo absoluto medio que comete la red:

$$MAPE = \sum_{i=1}^{n} \frac{y_i - \hat{y}_i}{y_i}$$

Este error puede ser considerado como un porcentaje de error con respecto al valor real. Cabe destacar que cuando algún valor real de la variable salida, no es posible calcular este error y el gráfico no sería adecuado, por lo que en dichos casos se muestra un mensaje

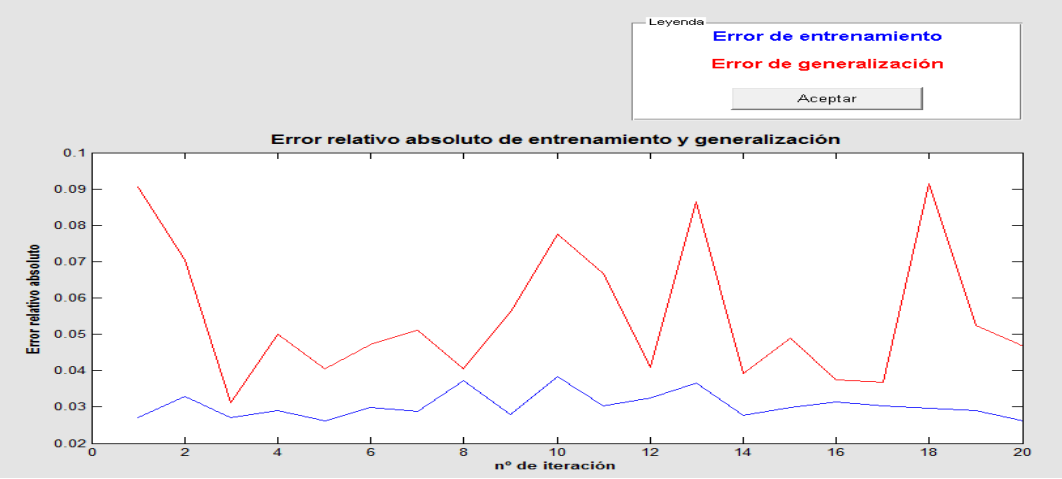

Figura 5.39: Gráfico del error relativo absoluto medio (MAPE)

de advertencia indicando que no se puede obtener dicho gráfico.

Al igual que en la anterior gráfica se representa una curva (azul) asociada al error en la fase de entrenamiento en cada iteración, y una curva (roja) asociada al error en la generalización por cada iteración. En la figura 5.39 se muestra un ejemplo de esta gráfica.

Gráfica Valores reales-Valores estimados en la fase de entrenamiento: Esta gráfica compara los valores reales de la variable salida frente a los valores estimados por la red, durante la fase de entrenamiento. En el caso de haber elegido *Hacer cross validation*, los datos graficados son los de la última iteración ejecutada.

Esta gráfica nos permite comprobar de un vistazo como de bien aproxima la red los datos de entrenamiento.

La figura 5.40 muestra un ejemplo de esta gráfica, donde la curva azul representa los valores de la variable salida estimados por la red, mientras que la curva roja representa los valores reales de dicha variable salida.

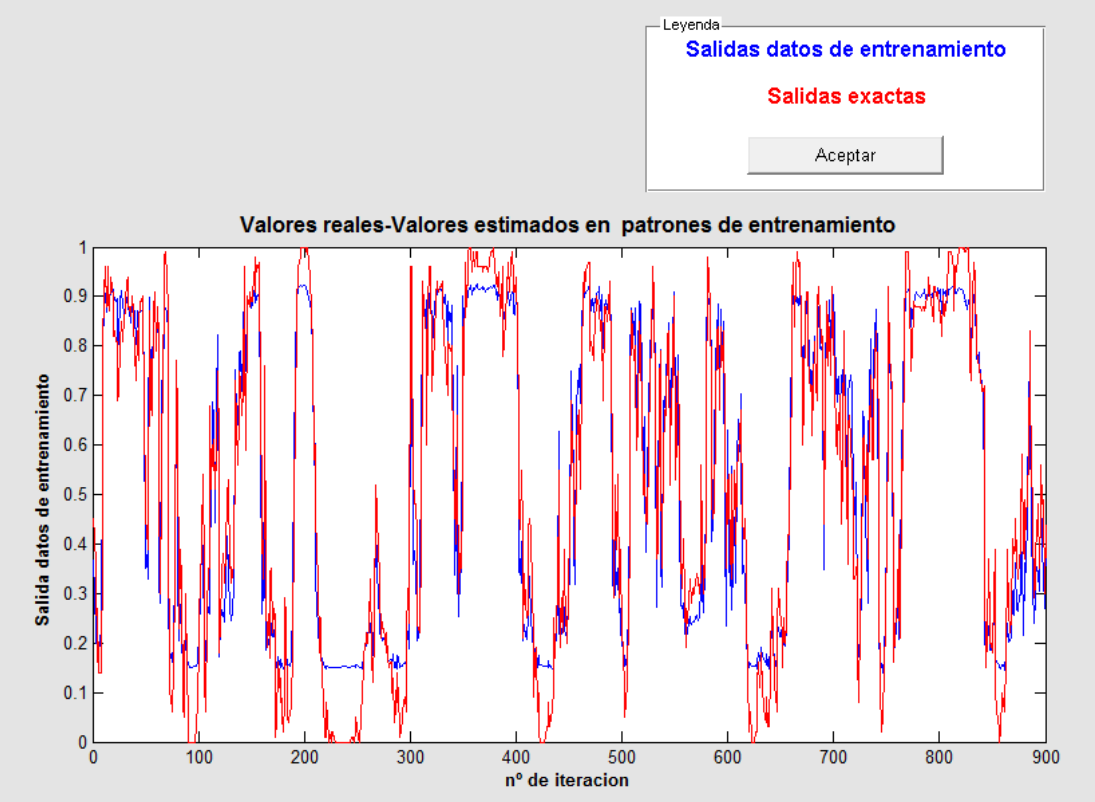

Figura 5.40: Gráfica Valores reales-Valores estimados en patrones de entrenamiento

Gráfica Valores reales-Valores estimados en la generalización: Esta gráfica es similar a la anterior, pero en este caso se representan los valores estimados y reales de la

variable salida, obtenidos durante la fase de generalización. En el caso de que se haya realizado un entrenamiento con validación externa, los datos graficados corresponden a los del fichero correspondiente. En caso haber realizado cross validation los datos que se toman son los datos de generalización correspondientes a la última iteración.

Al igual que la anterior gráfica, ésta le sirve al usuario para que de un simple vistazo compruebe como de bien generaliza la red, es decir, como de bien se comporta ante datos que en principio no conoce. Un ejemplo de este tipo de gráficas se muestra en la figura 5.41, donde la curva azul representa los valores estimados por la red, y la curva roja representa los valores reales.

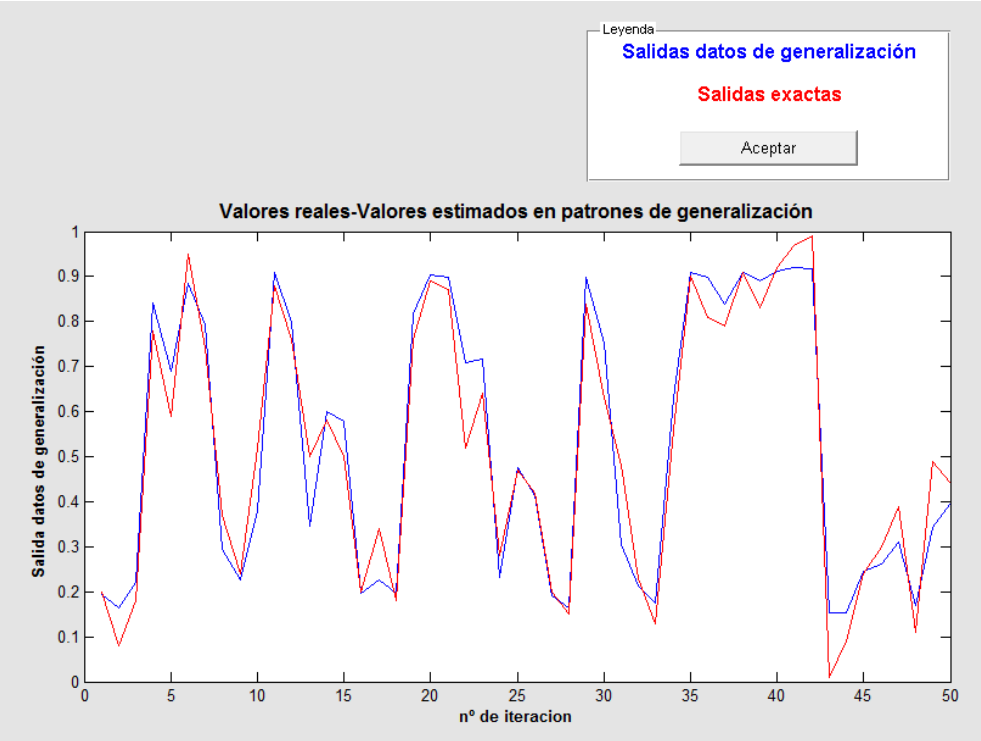

Figura 5.41: Gráfica Valores reales-Valores estimados en patrones de generalización.

Gráfica Valor estimado vs Residuos en la fase de entrenamiento: Es un gráfico de dispersión, una nube de puntos. En el eje X se representa el valor estimado por la red y en el eje Y, el residuo según el modelo estimado. Este gráfico permite ver si el error es un ruido blanco, con comportamiento aleatorio alrededor del cero. O si por el contrario, presenta, asimetrías (muchos valores positivos o negativos), heterocedasticidad (mayor dispersión para ciertos valores estimados de la variable), etc. Un ejemplo de este tipo de gráfico se muestra en la figura 5.41. Igual que en las opciones anteriores, si se ha realizado cross validation, los valores graficados corresponden a la última iteración.

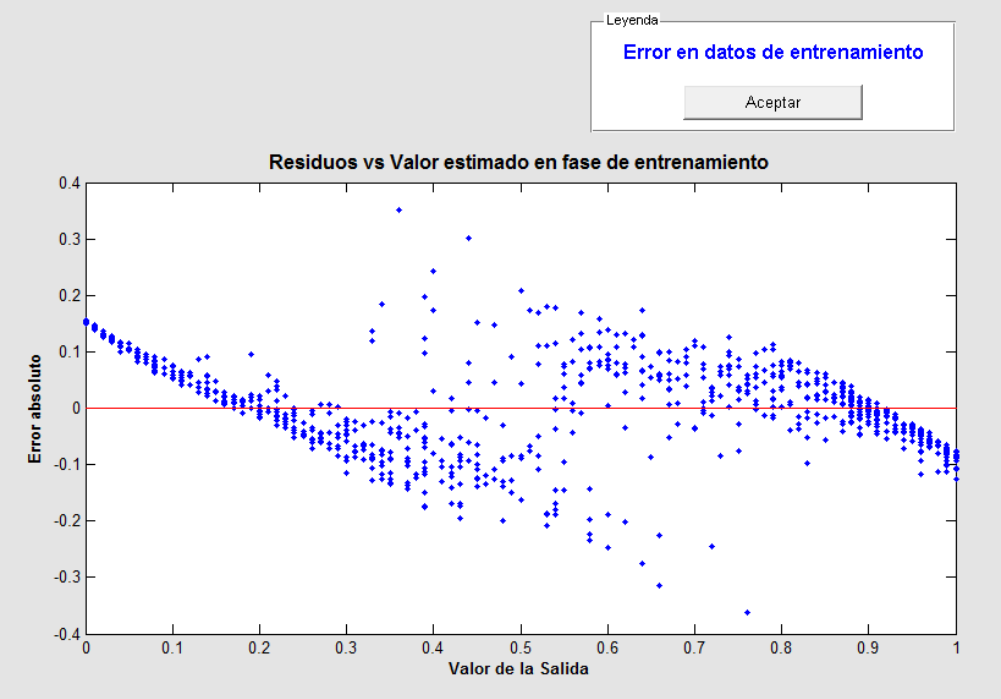

Figura 5.42: Gráfica Residuos vs Valor estimado en fase de entrenamiento.

Por ejemplo, en la gráfica ejemplo de la figura 5.42 podemos ver como la red ante salidas próximas a uno siempre tiene un error negativo y creciente, por lo que se puede pensar que la red elegida le "cuesta" que sus salidas alcancen valores igual a 1, y errores positivos cuando la salida es cercana a cero. Esto muestra que la arquitectura de la red no es adecuada.

Gráfica Valores estimados vs Residuos vs en fase de generalización: Está gráfica es similar a la anterior. La única diferencia es que trabaja con los datos usados en la fase de generalización. Estos datos pueden ser los pertenecientes al fichero que se ha elegido como fichero de validación externa o aquellos correspondientes a la última iteración en caso de haber entrenado utilizando la opción de cross validation. En caso de haber realizado entrenamiento simple esta gráfica no visualizará nada.

En la figura 5.43 se muestra un ejemplo de este tipo de gráficas. El usuario utilizará estas gráficas para cosas similares que en el caso anterior, pero además ésta, y a diferencia de la anterior, permite estudiar la capacidad de generalización de la red.

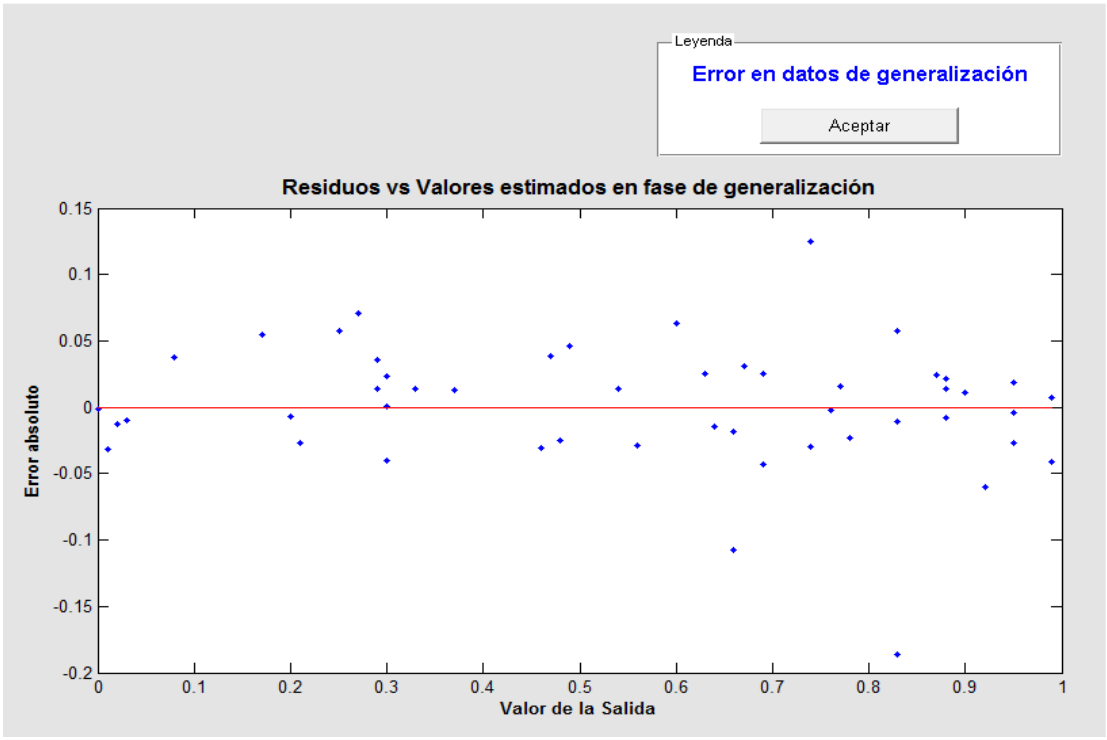

Figura 5.43: Gráfica Residuos vs Valor estimado en la generalización.

5.3.7.-Guardar estructura de red

Esta opción permite al usuario guardar la arquitectura de la red con el valor de los pesos de las conexiones sinápticas y umbrales para en otra ocasión cargarla y poder simular

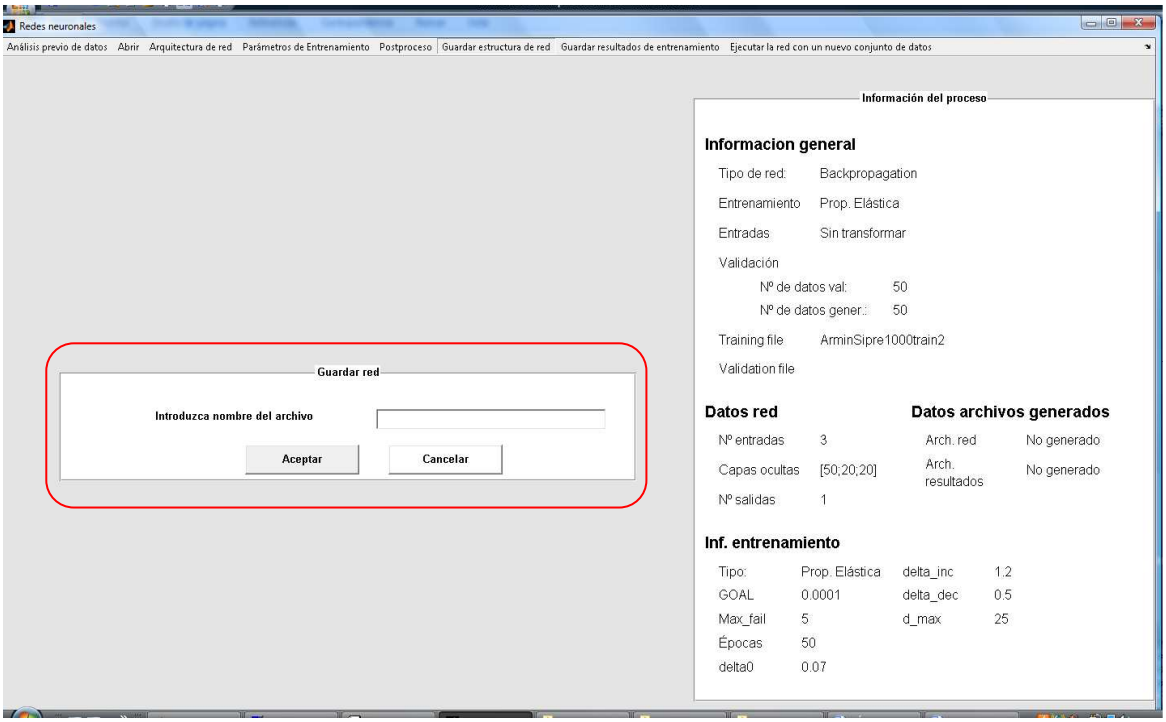

Figura 5.44: Panel de la opción *Guardar red*

nuevos datos con ella (mirar apartado 5.3.2.).

Cuando se selecciona esta herramienta el programa despliega un panel como el que se muestra en la figura 5.44. Dicho panel pide que se indique el nombre con el que se va a guardar la red. Una vez escrito se pulsa el botón *Aceptar* y el programa guarda la red en una nueva carpeta (si ésta no se ha creado previamente) llamada *red_creada*. Todas las redes que se guarden trabajando en este directorio se almacenan en dicha carpeta.

El programa entonces genera en el directorio con el que se esté trabajando, una carpeta denominada *red_creada* si previamente no estaba creada. El programa, dentro de dicha carpeta, genera cuatro archivos (formato *txt*) cuyos nombres son *Nombre de la red_biases*, *Nombre de la red_input*, *Nombre de la red_layer*, *Nombre de la red_minmax* y en algunos casos *Nombre de la red_trans,* donde *Nombre de la red* es el nombre que se le haya dado previamente. Cada archivo contiene la siguiente información y en el siguiente formato:

X_biases: Contiene el valor de los valores umbrales de cada una de las neuronas de cada capa. Éstos están agrupados en una columna, de forma que los primeros valores corresponden a los de la primera capa oculta, los siguientes, a la segunda capa oculta, y así sucesivamente.

X_input: Este archivo contiene más información. En la primera línea de dicho archivo aparece la información sobre el tipo de red, el número de neuronas de entrada y de la primera capa oculta. Los últimos tres números corresponde al tipo de transformación de las entradas que se ha aplicado durante el entrenamiento de la red. Sólo uno de los tres puede ser igual a 1. Si es el primero de ellos es que no se ha realizado transformación alguna, si es el segundo, las entradas se han escalado, y si es el tercero, se han estandarizado (mirar sección *Tratamiento de entradas* del apartado 5.3.4.). Seguidamente aparecen los valores de los pesos de las conexiones sinápticas de las neuronas de entrada con las de la primera capa oculta. Cada fila corresponde al valor de los pesos de las conexiones entre una neurona de la capa oculta con cada una de las entradas.

X_layer: La primera línea de este archivo describe totalmente la arquitectura de la red. Contiene información sobre el número de capas ocultas, las neuronas que tiene cada capa oculta y las salidas que tiene la red, y por último contiene información acerca de las funciones de activación que tiene cada capa, incluida la de salida. El valor uno representa la función *logsig*, el valor 2 representa la función *tansig*, y el valor 3 la función *purelin*.

Debajo de esta primera línea aparecen los valores de los pesos del resto de conexiones sinápticas que hasta ahora no se habían guardado (es decir todas salvo las que existen entre las neuronas de entrada y la primera capa oculta). Dichos valores están agrupados por capas.

X_minmax: Este archivo contiene los valores máximos y mínimos para cada entrada de los datos que se han utilizado durante el entrenamiento.

X_trans: Este archivo solo se crea cuando durante el entrenamiento de la red se ha utilizado alguna transformación en las entradas. Este archivo contiene la información necesaria para llevar a cabo la transformación de las nuevas entradas con las que se vayan a simular la red, y la pertinente transformación inversa para obtener las salidas. Esto permite al usuario despreocuparse de cómo se ha entrenado la red y si tiene que realizar alguna transformación a las entradas, ya que el programa lo realiza automáticamente.

Por otro lado es importante destacar que en caso de que falle alguno de estos archivos a la hora de cargar la red (mirar apartado 5.3.2.) el programa dará error. Además esta opción no se ha diseñado para obtener una visualización de resultados ya que al usuario le interesa bien poco el valor de los pesos de las conexiones, sino como una herramienta que nos permita volver a trabajar con redes ya entrenadas. Para guardar resultados importantes de la red y visualizarlo de una forma adecuada se ha diseñado la herramienta que se describe en el siguiente apartado.

5.3.8.-Guardar resultados de entrenamiento

Esta opción permite al usuario generar y guardar un archivo que contiene toda la información acerca de la arquitectura de la red, opciones de entrenamiento escogidas, tipo de entrenamiento (en su caso), resultados importantes de la fase de entrenamiento y

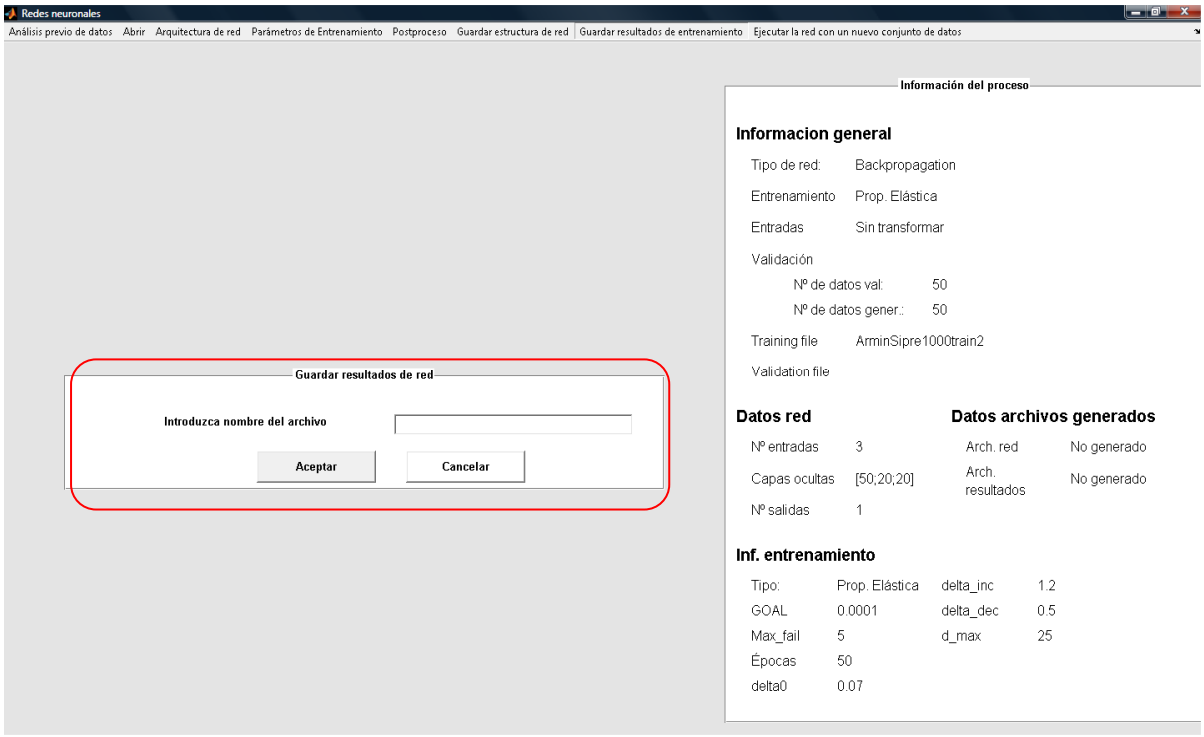

Figura 5.45: Panel de la herramienta *Guardar resultados de entrenamiento*

generalización de la red. A continuación se describirá más detalladamente la estructura de los posibles archivos que puede generar.

Para acceder a esta herramienta hay que pulsar en *Guardar resultados de entrenamiento* del menú de la parte superior del programa. A continuación el programa desplegará un panel que solicita que introduzcamos el nombre del archivo que se va a generar (figura 5.45).

Una vez introducido el nombre y pulsar el botón *Aceptar* se vuelve a desplegar un panel como el de la figura 5.46 que nos da la opción de abrir el archivo generado o de guardar y continuar. Si se pulsa la opción *Abrir archivo generado* se abrirá en el editor de textos de Matlab un fichero con toda la información arriba resumida.

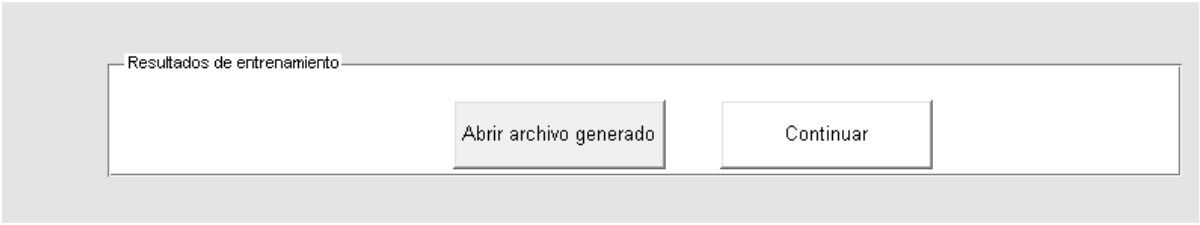

Figura 5.46: Herramienta *Guardar y ver resultados de entrenamientos.*

Cuando el programa genera y guarda este fichero, inicialmente genera una nueva carpeta en el directorio donde se esté trabajando que se llama *entrenamiento* (si previamente no existía). En esta carpeta se guardarán todos los ficheros relativos a los resultados del entrenamiento cuando se trabaje con el directorio que la contiene.

Dentro de esta carpeta se habrá guardado un documento de texto con el nombre que le hayamos dado previamente. Dicho fichero contiene la siguiente información:

En el caso de trabajar con redes backpropagation aparece primero el tipo de red con la que se está trabajando (en este caso red backpropagation). Seguidamente aparecerá información acerca de la arquitectura de la red; número de entradas, número de neuronas en las capas ocultas y número de salidas de la red. Después aparece información sobre el tipo de entrenamiento y valores de los parámetros que se han utilizado para dicho entrenamiento. Por último, en relación a la configuración del entrenamiento, se especifica si se ha realizado o no transformación, y en el caso de que sí, que tipo de transformación se ha empleado, la fórmula del error utilizado para la optimización de la red durante el entrenamiento, si se ha hecho o no cross validation y los ficheros de datos que se han utilizado durante el entrenamiento.

Posteriormente aparecen los resultados del entrenamiento. En primer lugar aparece el error cuadrático medio de entrenamiento que comete la red para cada iteración, y después aparece el error cuadrático medio para los datos de generalización para cada iteración,

ambos por salida de la red. Posteriormente aparece el error medio cuadrático de todas las iteraciones aplicándole la raíz cuadrada, tanto para los datos de entrenamiento como para los datos de generalización. Seguidamente (en el caso de utilizar datos de generalización) aparece el valor R2, que representa la variabilidad explicada por el modelo para cada variable de salida. Cuanto más próximo sea este valor a 1 mejor será el modelo estimado. Para su cálculo se ha empleado el error cuadrático medio (media de todas las iteraciones) y la varianza de la variable de salida (datos de generalización) según la expresión:

$$1 - \frac{mse\ medio}{Var\ (Y1)}$$

Finalmente aparece, para la última iteración, los valores de los datos de entrenamiento y de generalización en caso de haberlos utilizado.

En el caso de trabajar con redes de base radial existen dos diferencias en relación a lo explicado anteriormente. Una de ellas es que en la información relativa a la arquitectura se muestran datos acerca del número de entradas y salidas, y el máximo número de neuronas que se han especificado para dicha red. La otra es que en el caso de poder trabajar con varios spreads, se muestran los errores cometidos por la red para cada spread. Sin embargo las simulaciones de la red con los datos de entrenamiento y de generalización corresponden al último spread.

5.3.9.-Ejecutar la red con un nuevo conjunto de datos.

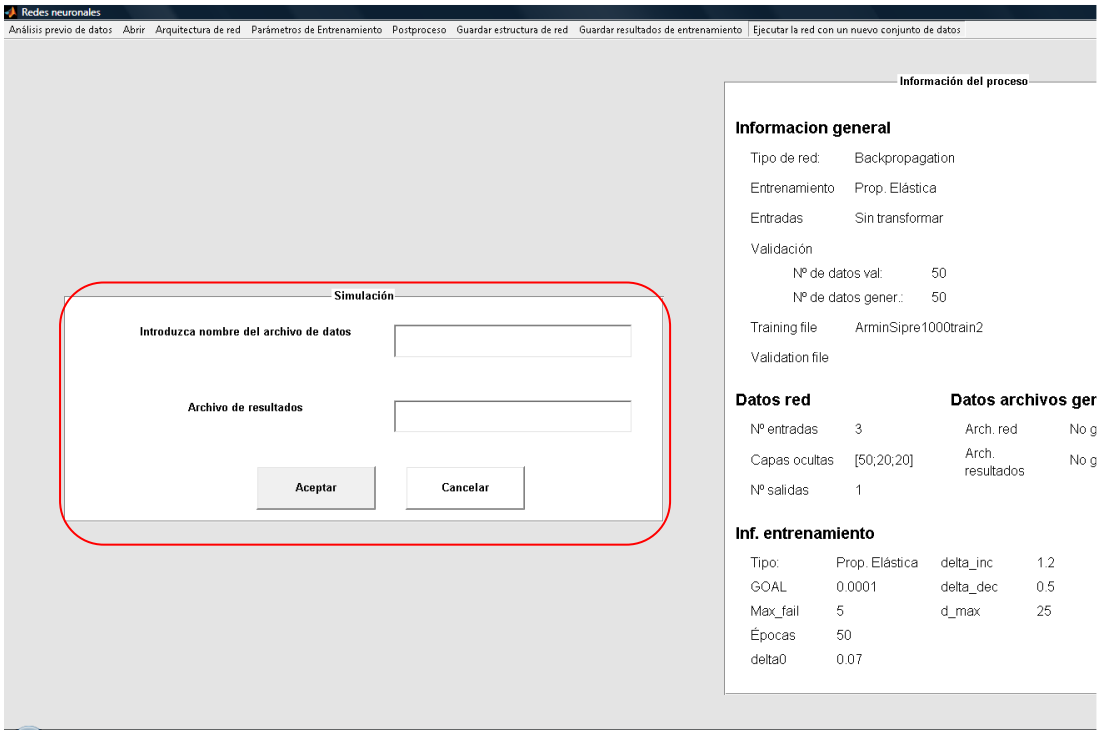

Figura 5.47: Panel para especificar el nombre de los archivos de datos de simulación y resultados

Esta herramienta permite simular la red usando nuevos valores de las variables entradas, los cuales deben ser introducidos a través de un fichero. Dicho fichero debe ser un documento de texto y tener una estructura similar a los ficheros de entrenamiento pero sin tener en este caso las columnas correspondientes a los valores de las variables de salida. Al igual que ocurría en el resto de ficheros debe encontrarse en el directorio de trabajo.

Cuando se accede a esta funcionalidad del programa aparecerá una ventana como la que se muestra en la figura 5.47. En ella hay que introducir el nombre del archivo que contiene los datos que se van a simular y el nombre que se desee para el archivo de resultados.

Cuando se pulsa el botón *Aceptar* vuelve a desplegarse un panel que nos da la opción de abrir el archivo generado o continuar, similar al de la figura 5.46. El archivo generado se guarda en una carpeta llamada simulaciones la cual se encuentra en el directorio de trabajo. Dicho archivo contiene cada dato (es decir las entradas a la red) que se ha simulado en un fila acompañado de las respectivas salidas que haya dado la red.

5.3.10.-Ayuda.

En este menú encontraremos las ayudas relativas a cada módulo anteriormente descrito, así como una ayuda que resalta temas importantes que debe tener en cuenta el usuario (Aspectos generales). Dichas ayudas deben encontrarse en el directorio donde se almacenan los ficheros de la GUI en una carpeta llamada *Ayuda general*.

Asimismo, algunos paneles disponen de un título en color azul lo cual indica que dicho panel dispone de una ayuda específica. Las ayudas específicas se encuentran en otra carpeta (dentro del mismo directorio anterior) llamada *Ayudas específicas*.

Estas ayudas se han diseñado teniendo en cuenta que el usuario no tiene amplias nociones sobre redes neuronales, de forma que le ayuden a utilizar el programa de manera sencilla y entendiendo lo que se está realizando.

5.4.-Panel de listado de errores.

Como se comentó en el apartado 5.1 el panel de listado de errores se encuentra en la zona marcada como 2 según la figura 5.1. Cuando se ha seleccionada la arquitectura de la red y se ha definido el entrenamiento se puede proceder a entrenar a la red. Cuando el usuario ejecuta la opción de *entrenar* (mirar apartado 5.3.5.) el programa primero realiza unas comprobaciones para verificar que lo que se ha definido en el entrenamiento es coherente con la arquitectura de la red y el significado propio de los parámetros, de forma que si el

programa detecta que el usuario ha especificado algo mal, detiene el entrenamiento y muestra un mensaje en dicho panel describiendo lo que sucede.

El programa es capaz de detectar si el usuario no ha especificado ningún tipo de red o ningún tipo de entrenamiento, si algún valor que no puede ser negativo es negativo, si la suma de las entradas y salidas especificadas por el usuario no coinciden con los datos pasados por el fichero, y por último si se quieren más datos de validación y generalización de los que hay en los ficheros de datos de entrenamiento que se utiliza.

5.5.-Panel de información del proceso.

En la figura 5.48 se muestra donde aparece este panel y lo que inicialmente contiene. La funcionalidad de este panel es la de mostrar información sobre la arquitectura de la red y el entrenamiento escogido. En este panel se distinguen cuatro grandes bloques de información (ver figura 5.45).

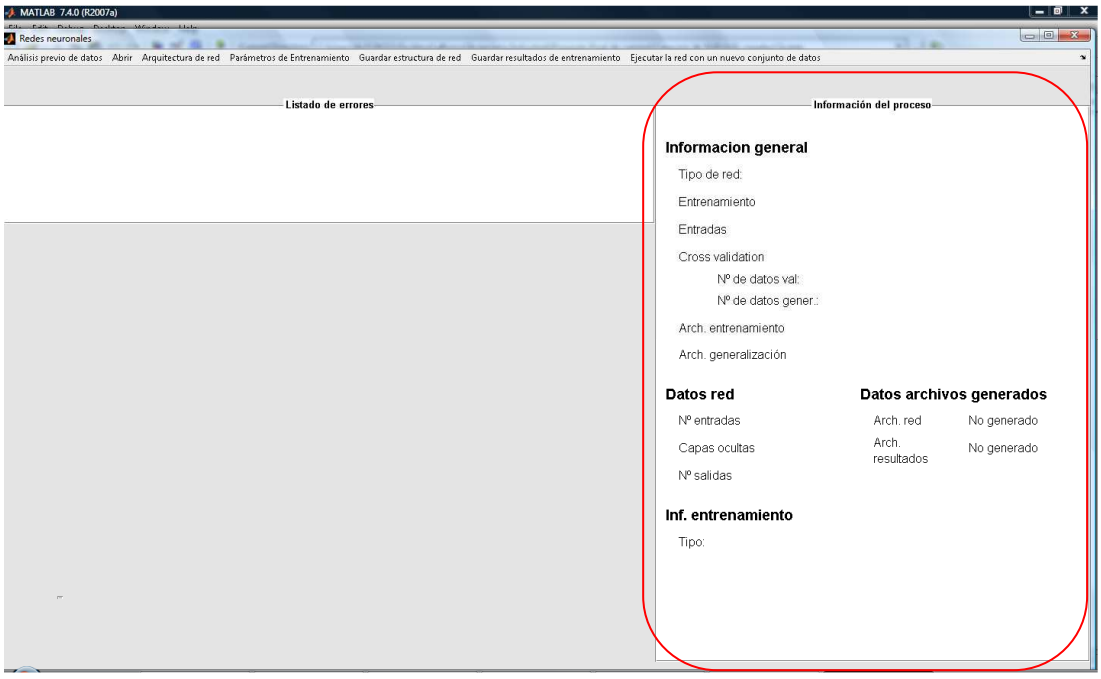

Figura 5.48: Panel de información del proceso

El primer bloque se muestra información general acerca del entrenamiento y el tipo de red, distinguiéndose los siguientes campos:

Tipo de red: En este campo aparece si la red es backpropagation o de base radial

Entrenamiento: En el caso de utilizar redes backpropagation en este campo se especifica el tipo de entrenamiento que se va a utilizar entre los descritos en el apartado 5.3.5. Si se utilizan redes de base radial este campo está vacío.

Entradas: En este campo se especifica el tipo de transformación que se va a utilizar. Si aparece vacío quiere decir que se utilizará la opción por defecto, es decir, no realizar ninguna transformación.

Cross validation: Contiene dos campos. En el primero de ellos se especifica el número datos de validación, y en el segundo, el número de datos de generalización.

Arch. Entrenamiento: En este campo se especifica el nombre del archivo que contiene los datos de entrenamiento que se van a utilizar.

Arch. Generalización: En este campo se especifica el nombre del archivo que contiene los datos de generalización en caso de utilizarse generalización externa.

En el segundo bloque se muestran datos acerca de la arquitectura de la red escogida. Como existen dos tipos de redes, los nombres de los campos de este bloque cambiarán dependiendo de si se ha escogido una o la otra. En el caso de que se escoja una red backpropagation éstos son los campos.

Nº de entradas: Especifica el número de entradas de la red.

Capas ocultas: Muestra un array de números donde cada número representa el número de neuronas de su respectiva capa oculta. Están puestos en orden.

Nº de salidas: Muestra el número de salidas que tiene la red.

En el caso de utilizar redes de base radial, los campos primero y tercero permanecen iguales al caso mostrado anteriormente. El segundo cambia a:

Spread: Existen dos posibilidades en este campo. Si en la arquitectura de la base radial se ha escogido trabajar con un único valor de spread, en este campo se mostrará este valor. Si por el contrario se ha decidido trabajar con varios valores de spread en este campo se muestra el intervalo de trabajo.

En el tercer bloque de información aparece información relativa al valor de los parámetros del algoritmo de optimización escogido para la red, que tiene que ser backpropagation. Por ello este bloque no significa nada en el caso de que se esté trabajando con redes de base radial.

Por último el cuarto bloque de información, *Datos archivos generados*, muestra si se ha generado o no los archivos de estructura de la red o de resultados de entrenamiento para esta red.

5.6.-Comparativa con herramientas similares de Matlab.

5.6.1.-Introducción.

En este apartado se comentarán a modo de resumen las características de GUI que tiene Matlab para el entrenamiento de redes neuronales, además de describirse sin mucho detalle el funcionamiento de dicha herramienta.

Por último se realizará una comparativa entre esta herramienta y la desarrollada en este proyecto, enumerando las bondades de uno y de otro. Del mismo modo, y aprovechando dicha comparativa, se justificará el uso de esta herramienta frente a la que contiene Matlab para resolver problemas de ingeniería.

5.6.2.-Descripción de la GUI *nntool* de Matlab.

Para acceder a esta herramienta en Matlab es necesario introducir en la línea de comandos de Matlab la orden *nntool*. Posteriormente se abre una ventana como la que se muestra en la figura 5.49.

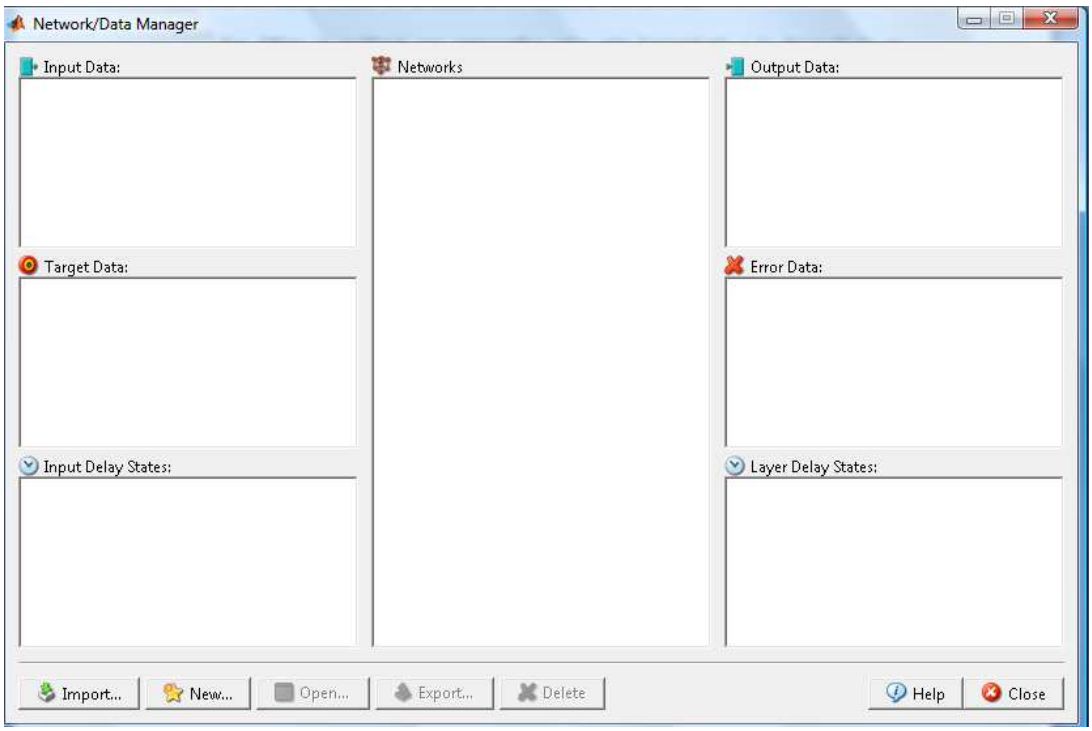

Figura 5.49: Herramienta para el entrenamiento de redes neuronales de Matlab

Como se ve en la figura, en ella aparecen siete campos. Cada campo almacena un determinado tipo de objeto. Así el campo *Networks* almacena la arquitectura de la red, el campo *Input data* almacena variables que contienen entradas, el campo *Target data*,

contiene variables que almacenan las salidas objetivo que se utilizarán durante el entrenamiento, en el campo *Output data*, las salidas de una red creada cuando se simulan variables almacenadas en el campo *Input data*. El resto de campos por ahora no nos interesan porque no se utilizan en nuestro proyecto.

Para crear un elemento nuevo en alguno de los campos, se pulsa el botón *New* que se encuentra en la zona inferior. Cuando se pulsa dicho botón se abre una nueva ventana, que se muestra en la figura 5.50.

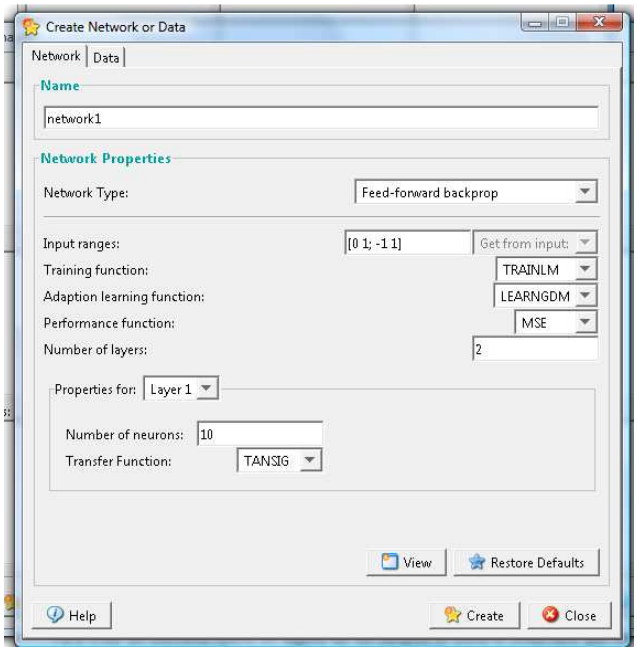

Figura 5.50: Cuadro de diálogo para la creación de redes

Este cuadro de diálogo tiene dos pestañas (mirar parte superior), una para crear redes llamada *Network*, otra para crear matrices de datos llamada *Data*, los cuales se utilizarán como entradas (en el campo *Input data*) o como objetivos para el entrenamiento de la red (en el campo *Targets*).

Como vemos en la figura 5.50 para la creación de la red hay que especificar los siguientes campos: El tipo de red que se va a utilizar y el rango de las entradas (para ello se puede utilizar alguna variable que se encuentre en el campo *Input data*). Dependiendo del tipo de red que se escoja lo siguiente puede variar. Si se escoge redes backpropagation aparece la función de entrenamiento (que depende del tipo de red que se va a utilizar pero es análogo al algoritmo de optimización utilizado en el programa), el tipo de error a optimizar, el número de capas ocultas y las neuronas de éstas capas (todas las capas tienen el mismo número de neuronas). Si se escogen redes de base radial, únicamente hay que elegir el spread, las entradas y salidas.

Cuando se haya pulsado el botón *Create* se crea un nuevo objeto en el campo *Network*.

Para crear los datos de entrenamiento de entrada y salida la aplicación da dos opciones. Una de ellas, que pocas veces se utilizará es la de introducir los datos a mano. Para ello hay que pulsar el botón *New* de la ventana principal. La otra es importarlo del espacio de trabajo de Matlab o de un archivo .M que se encuentre en el directorio de trabajo de Matlab. Para ello hay que pulsar el botón *Import* de la ventana principal. Cuando se hace esto aparece la ventana que se muestra en la figura 5.51.

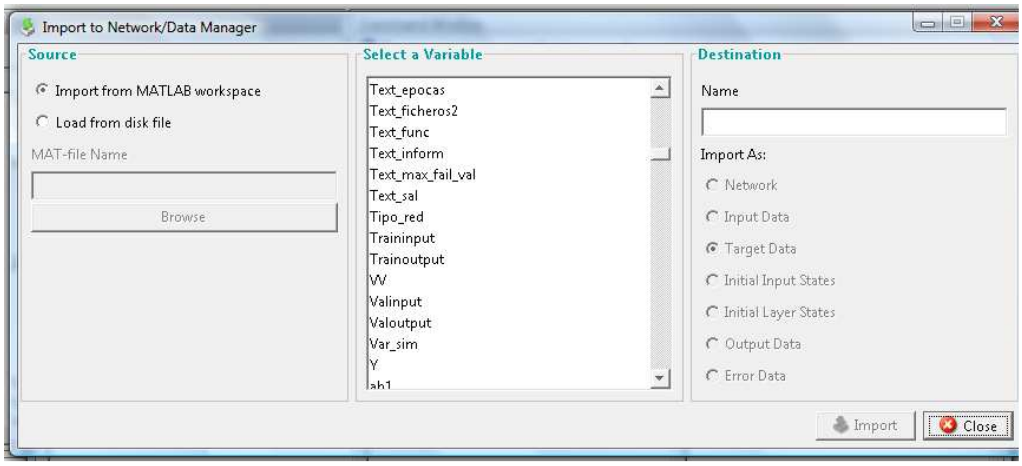

Figura 5.51: Cuadro de diálogo para importar datos.

En este panel hay que seleccionar la variable deseada en el campo *Select a Variable* e indicar a que campo de la ventana principal lo quieres importar, si al de entradas, salidas, etc. Inicialmente habrá que seleccionar si importamos del espacio de trabajo de Matlab o de un archivo externo.

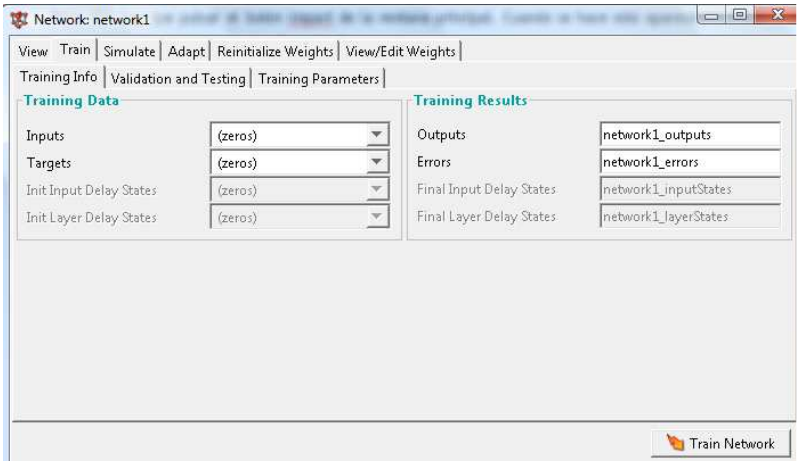

Figura 5.52: Opciones para trabajar con redes

Una vez creado todos los objetos, para poder entrenar la red es necesario abrir dicha red. Para ello se selecciona la red deseada del campo *Network* y se pulsa el botón *Open* que se encuentra en la parte inferior. Posteriormente aparecerá algo similar a lo que aparece en la figura 5.52. En esta ventana aparecen todas las herramientas de la que dispone el programa para trabajar con redes.

En dicha ventana aparecen diversas pestañas. Para entrenar la red hay que encontrarse en la pestaña *Train*. En dicha pestaña aparecen todas las opciones para especificar el entrenamiento, datos de entrada, objetivos, parámetros de entrenamiento, etc. Cuando todo esté seleccionado se pulsa el botón *Train Network*.

También se pueden simular redes en la pestaña *Simulate*, donde hay que especificar las entradas y en que variable queremos almacenar las salidas. Además se puede ver la estructura de la red, inicializar los pesos, etc.

5.6.3.-Comparativa.

En este apartado se realizará una comparativa entre la aplicación que posee Matlab y la desarrollada en este proyecto.

En primer lugar cabe destacar que la aplicación desarrollada aquí se ha centrado exclusivamente en dos tipos de redes, las redes bakckpropagation y las redes de base radial por ser éstas las más utilizadas para la resolución de problemas de ingeniería. Y esto precisamente es una diferencia existente entre estas dos aplicaciones, ya que la aplicación de Matlab contiene gran número de distintas redes que se pueden generar.

Sin embargo la manera de cargar datos de entrenamiento y validación en la aplicación de Matlab es bastante rudimentaria, teniendo que haberla cargado antes en el espacio de Matlab y posteriormente importarla al programa, lo que obliga al usuario a conocer aspectos básicos de Matlab. Además ésta es bastante rígida a la hora de definir la arquitectura de la red bakpropagation ya que no se puede especificar diferente número de neuronas en cada capa oculta. Otro aspecto positivo a favor de la aplicación desarrollada en este proyecto es que da posibilidad al usuario de realizar transformaciones a los datos con el fin de mejorar el entrenamiento de la red.

Por otro lado, la aplicación de Matlab no tiene la opción de realizar *Cross Validation* qué es una herramienta muy útil para obtener unos parámetros adecuados para conocer la bondad del comportamiento de la red. Asimismo, no dispone en la misma GUI de una herramienta para hacer un análisis estadístico de los datos, que si ha sido implementada en esta GUI y que puede resultar de mucha utilidad para los usuarios, antes de empezar con la modelización de redes neuronales.

Por último y más importante para una aplicación de este estilo es la visualización de resultados. La visualización de resultados de la aplicación de Matlab es bastante deficiente por no decir inexistente, sin embargo la aplicación desarrollada en este proyecto tiene numerosas posibilidades para ver distintos resultados de forma que ayuden al usuario a optimizar la red. Además la GUI ofrece la posibilidad de generar de forma automática una serie de ficheros que contienen toda la información necesaria para volver a reproducir la red creada, o a partir de los resultados obtenidos obtener otros parámetros de error o medidas de ajuste del modelo estimado o realizar otro análisis de interés para el usuario.

5.7.- Compatibilidad con distintas versiones de Matlab.

Esta aplicación ha sido desarrollada con la versión 7.4 de Matlab, por lo tanto se asegura el funcionamiento de la GUI dentro de esta plataforma. También se ha comprobado que en la versión 7.3 funciona correctamente. Sin embargo en la versión 7.0 de Matlab la GUI no se puede utilizar porque Matlab da un error. No se ha podido comprobar el por qué por no disponer el autor de dicha versión.

Por otro lado, en la versión 8.x de Matlab ha sido necesario eliminar la posibilidad de utilizar datos de validación durante el entrenamiento de redes backpropagation, ya que en dicha versión de Matlab esta posibilidad se ha eliminado. Por ello, en este proyecto existen dos versiones de la misma GUI, una compatible a partir de la versión 7.3, y otra compatible con la versión 8 de Matlab.

Capítulo 6: Aplicaciones

6.1.- Diseño y entrenamiento de una red neuronal para la predicción de fuerzas de corte en una operación de taladrado.

6.1.1.-Introducción.

El problema que se intenta resolver es conseguir un modelo que nos permita predecir la fuerza de corte y la fuerza axial en una operación de taladrado a través del conocimiento de ciertas variables que caracterizan el mecanizado. Estas variables son, el tipo de herramienta, el tipo de taladro a realizar, material a mecanizar, recubrimiento de la herramienta, redondeo, flanco, geometría de filo recrecido, geometría de cráter, los ángulos $\alpha, \beta \, y \, \gamma$, de la herramienta, velocidad de corte, avance y profundidad. Se dispone de un total de 150 datos.

A la vista de los datos que disponemos, en los que muchas variables son iguales para todos los casos o unas dependen directamente de otras (hay correlación entre ellas), en el modelo se emplearán 8 variables de entrada: redondeo, flanco, una dimensión de la geometría del recrecido del filo, otra del cráter, los ángulos α y γ, la velocidad de corte y el avance. Por otro lado, el modelo tendrá dos salidas: fuerza de corte y fuerza axial.

6.1.2.-Análisis previo de datos

Inicialmente hay que estudiar los datos de los cuales disponemos para llevar a cabo el entrenamiento. Para ello utilizamos las herramientas del análisis previo de datos. Pero inicialmente necesitamos cargar el fichero de datos, que se llama *DatosFuerzas* (fijarse en la figura 6.1).

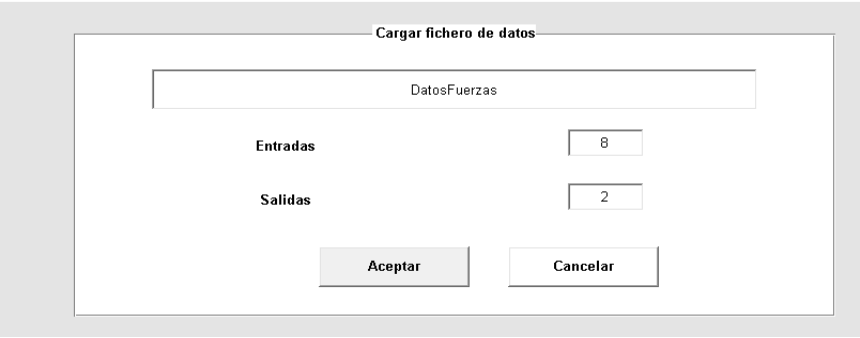

Figura 6.1: Cargar fichero de datos para su análisis previo

Posteriormente hay que seleccionar las columnas que son entradas y las columnas que son salidas (mirar figura 6.2).

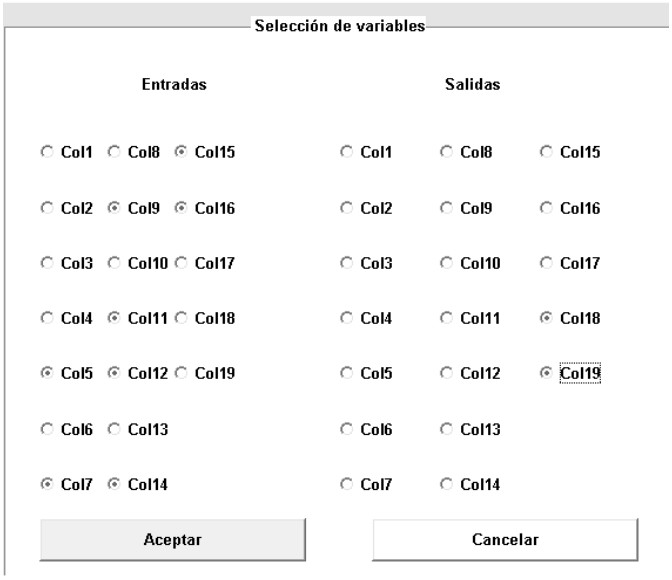

Figura 6.2: Selección de variables

A continuación estudiamos estos datos. Cabe destacar que el orden de las variables de entrada y de salida es el mismo orden que se ha establecido a la hora de seleccionar las columnas. Por ejemplo, si la col 3 del fichero es la primera columna de entrada, esa variable será la primera variable de entrada del modelo.

En primer lugar vamos a obtener ciertos parámetros estadísticos utilizando la herramienta del mismo nombre. En la figura 6.3 se muestran los resultados sacados por el programa. Para este caso no podemos sacar gran información, salvo los valores en los que se mueven las salidas, ver que ambas tienen valores del mismo orden; que únicamente disponemos de datos para dos velocidades de corte y dos avances distintos; que para la tercera variable (cráter) la mayoría de datos que disponemos son cero y poco más.

Parámetros estadísticos de las variables

	Media	Desv. Típica	Máximo	Mínimo	Percentil 20	Percentil 80
X1	0.03557	0.0338	0.15	0	0.02	0.071
X2	0.1855	0.27643	0.8	0	0	0.4
X3	0.014094	0.046528	0.2	0	0	0
X4	0.061745	0.19577	0.8	0	0	0
X5	0.096771	0.021688	0.1461	0.0873	0.0873	0.0873
X6	1.3946	0.0856	1.5294	1.2851	1.309	1.4835
X7	183.6242	60.0924	240	120	120	240
X8	0.075168	0.025084	0.1	0.05	0.05	0.1
Y1	353.9812	96.7752	573	174.4	255.59	442.06
Y2	274.0114	105.283	688.4	76.5	192.65	344.38

Figura 6.3: Resultados de la herramienta *Parámetros estadísticos*

Para conocer mejor los datos es útil obtener los diferentes histogramas de las variables. Para ello utilizamos la herramienta *Histograma de las variables*. Cuando se ejecuta dicha funcionalidad aparece un panel similar al de la figura 5.10. En él se selecciona cada una de las variables para obtener su histograma. En las siguientes figuras se representa los histogramas correspondientes a cada una de las variables que tiene el problema.

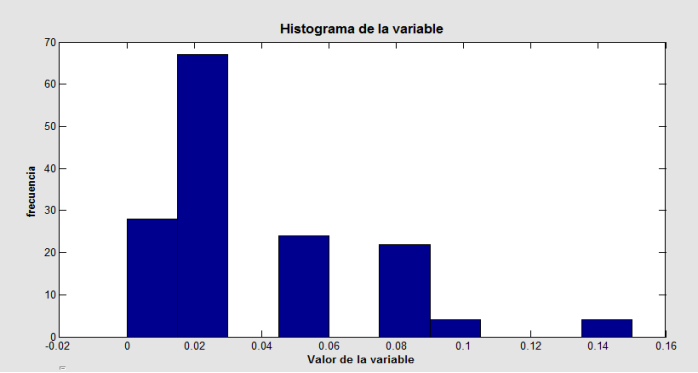

Figura 6.4: Histograma variable X_1- Redondeo

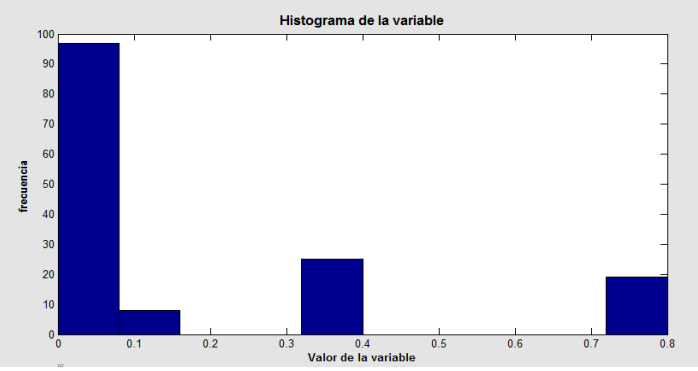

Figura 6.5: Histograma variable X_2- Flanco

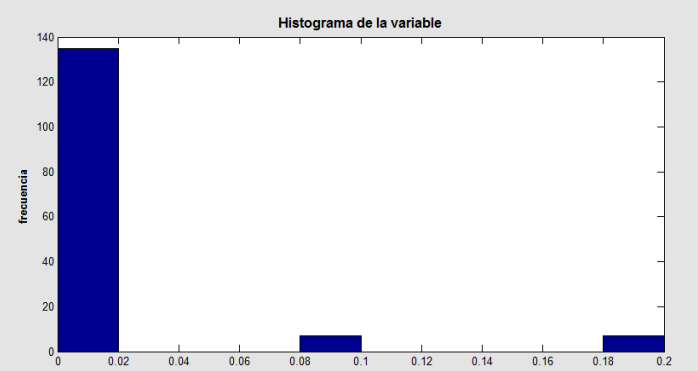

Figura 6.6: Histograma variable X_3- Recrecido

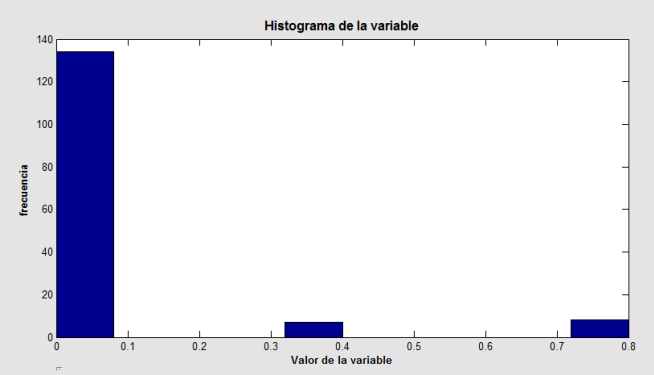

Figura 6.7: Histograma variable X_4- Cráter

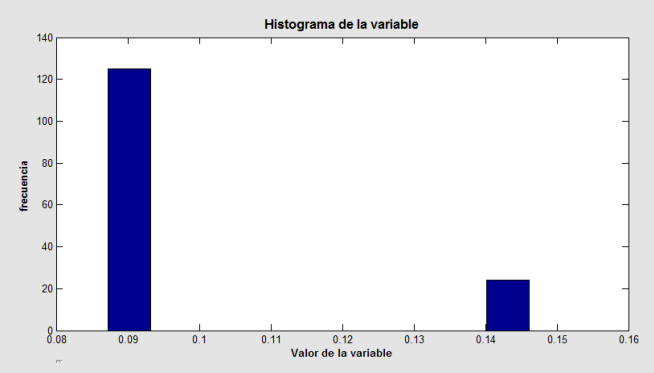

Figura 6.8: Histograma variable X_5- Ángulo α

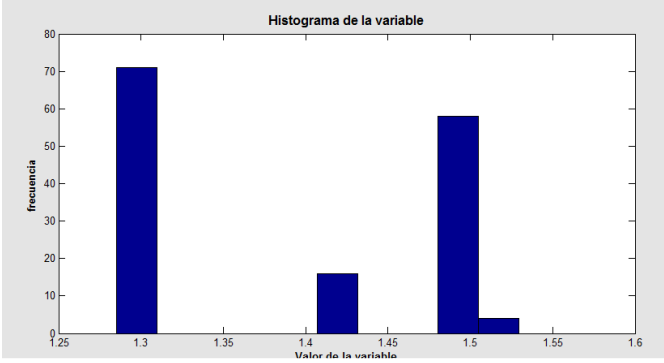

Figura 6.9: Histograma variable X_6- Ángulo γ

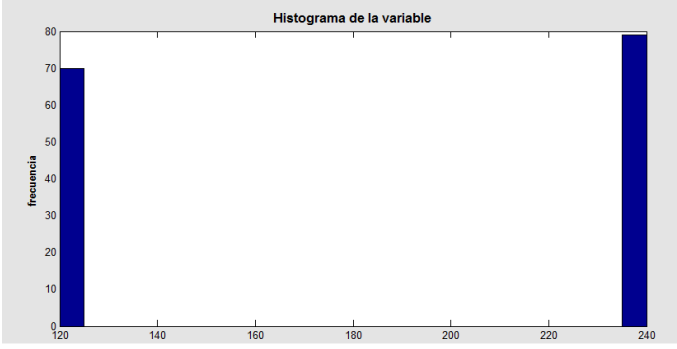

Figura 6.10: Histograma variable X_7- Velocidad de corte

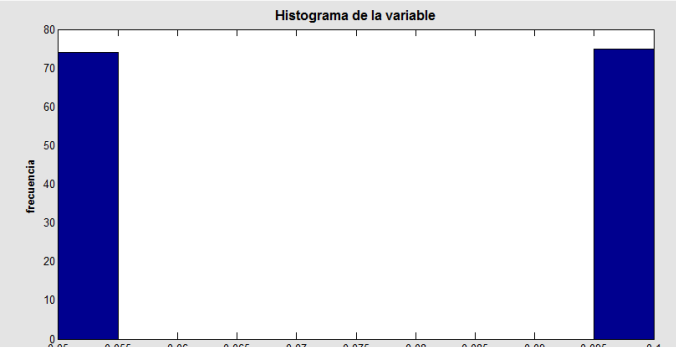

Figura 6.10: Histograma variable X_8- Avance

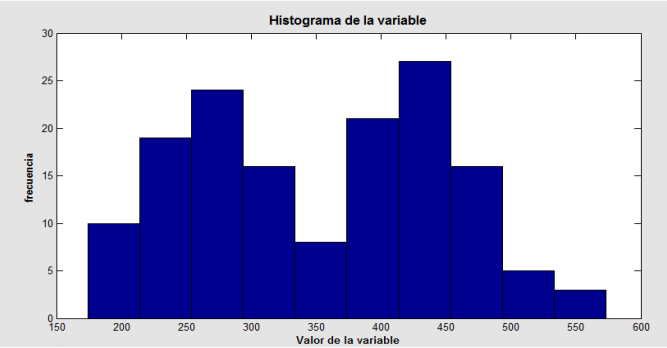

Figura 6.11: Histograma variable Y_1- Fuerza de corte

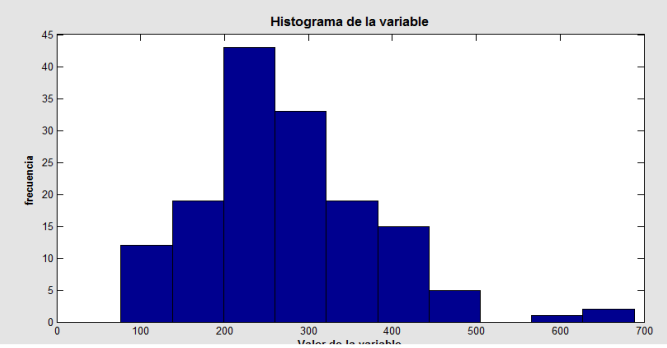

Figura 6.12: Histograma variable Y_2- Fuerza axial

Como vemos en los histogramas, las entradas no siguen una distribución continua, sino que únicamente toman ciertos valores, son distribuciones discretas. Esto viene determinado por la forma de obtener los datos (de forma experimental, donde los parámetros de entrada están controlados), pero a la vez esta estructura de datos va a ser determinante a la hora de elegir el tipo de red como se verá posteriormente. En cuanto a las variables de salida, la fuerza de corte tiene una distribución bimodal, que debe producirse por los valores de la velocidad de corte. La fuerza axial presenta unos algunos valores muy alejados del resto, podrían ser datos erróneos, sería interesante saber porque se presenta esa discontinuidad en el histograma.

Seguidamente deberíamos visualizar la manera en que se relacionan cada una de las variables de entrada con cada una de las variables de salida, sin embargo, debido a que los datos que tenemos de las entradas sólo toman ciertos valores, las gráficas que se visualizarían

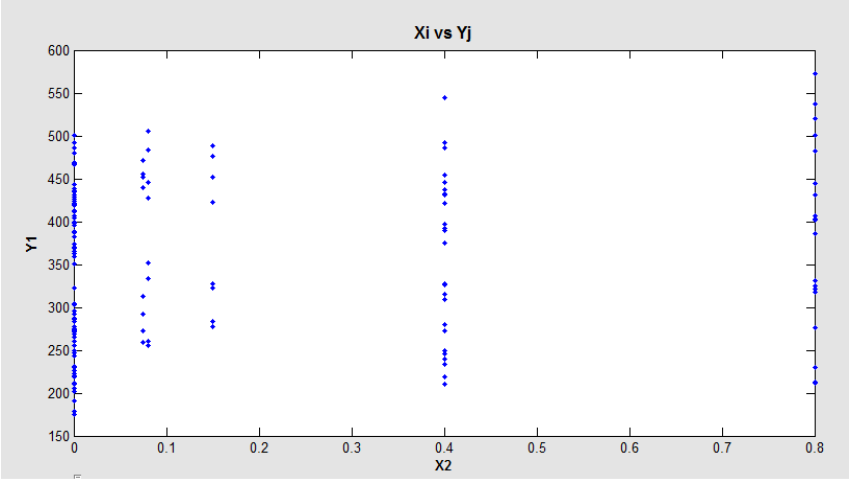

Figura 6.13: Gráfico de X_3 vs Y_1

serían parecidas a la de la figura 6.13, lo cual no nos sirve de gran ayuda.

Lo mismo ocurre con la herramienta *Boxplot* para las entradas, donde no obtendríamos información útil. Por otro lado, en la figura 6.14 se muestra el boxplot para las salidas que nos ayuda a visualizar en que rangos de valores se mueven y como es una respecto a la otra. Para obtener el gráfico de la figura 6.14 únicamente ejecute la herramienta *Boxplot de las salidas*.

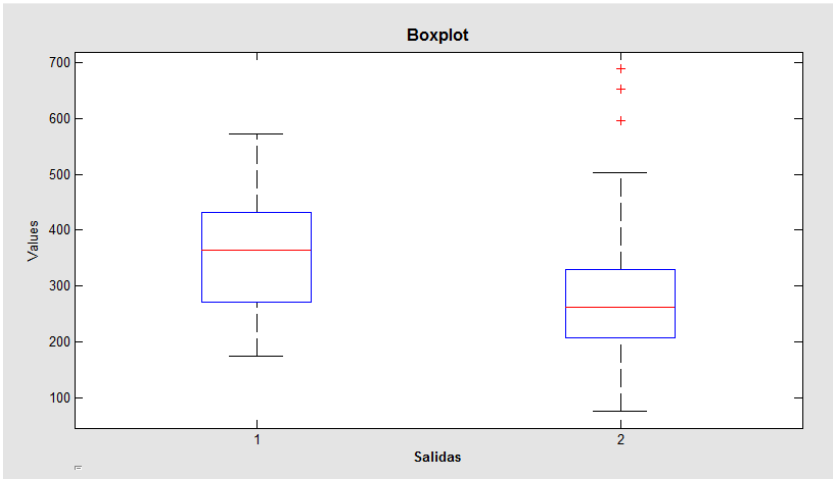

Figura 6.14: Boxplot de las salidas

En el gráfico el primer boxplots corresponde a la fuerza de corte y nos indica que esta variable sigue una distribución bastante simétrica, no se presentan valores alejados de la mayoría. En el caso de la fuerza axial (segundo boxplots) se ve una distribución más asimétrica y vuelven a aparecer un conjunto de valores alejados del resto de la distribución. (las tres cruces de arriba en la salida 2). Vemos también que en ambas existe una alta concentración de datos en torno a la media, pero que después su máximo y mínimo están bastante alejados, por lo que se espera que para valores en torno a la media la red que se entrene aproxime bien, ya que dispone de mayor cantidad de datos en esta región.

Por último, vamos a utilizar la última herramienta del módulo, *Modelo de regresión lineal*. Esta la utilizaremos para obtener un modelo de regresión lineal múltiple que nos permitirá descartar o no que el modelo no es lineal. Hay que tener en cuenta que si el modelo lineal fuese adecuado, la estimación obtenida mediante regresión lineal múltiple resultaría mejor que un modelo obtenido con redes neuronales. Si el modelo no es lineal, este análisis servirá, por un lado, para identificar posibles datos erróneos, y por otro lado obtener una posible referencia que nos permita comparar las bondades del modelo de redes neuronales.

En primer lugar puede ser conveniente obtener las ecuaciones lineales que relacionen las salidas con las entradas. Para ello se ejecutará la herramienta *Estimar modelo*. La aplicación nos mostrará los resultados de forma similar a como aparece en la figura 6.15.

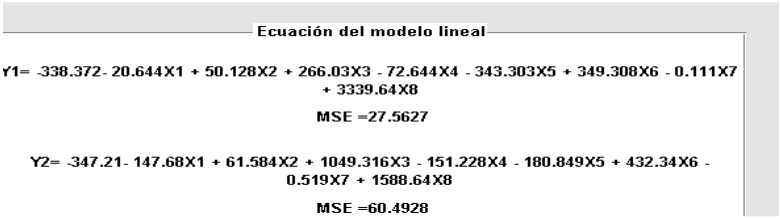

Figura 6.15: Ecuaciones del modelo lineal

Asimismo, y como se ve en la figura, para cada variable muestra la raíz del error cuadrático medio del modelo lineal.

Posteriormente es importante realizar algunos análisis para ver el grado de ajuste del modelo, ver si es adecuado o no. Podríamos obtener un histograma de los residuos que nos ayude a ver qué errores comete el modelo lineal y como se distribuyen. Para ello ejecutamos la herramienta *Histograma de los residuos*. En primer lugar aparece un panel similar al de la figura 5.15 donde tenemos que elegir respecto de que variable de salida queremos obtener los residuos, y cuantas clases se quiere que tenga el histograma. Una vez elegido obtenemos los siguientes dos histogramas, uno por cada variable.

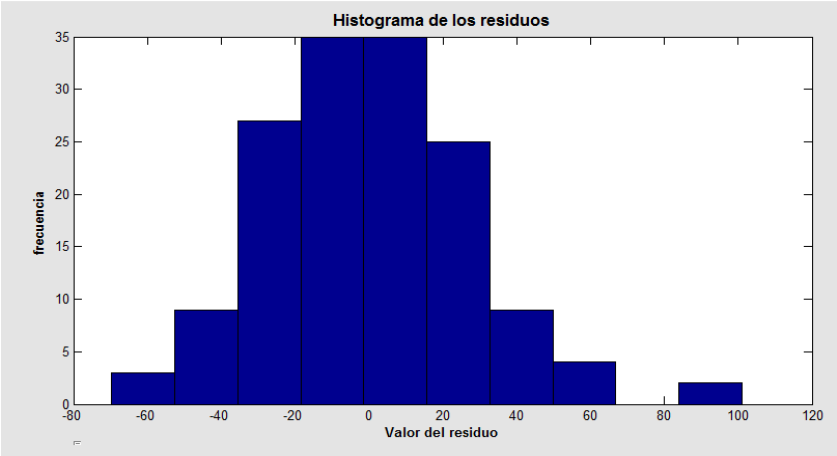

Figura 6.16: Histograma de residuos para la variable Y_1-Fuerza de corte

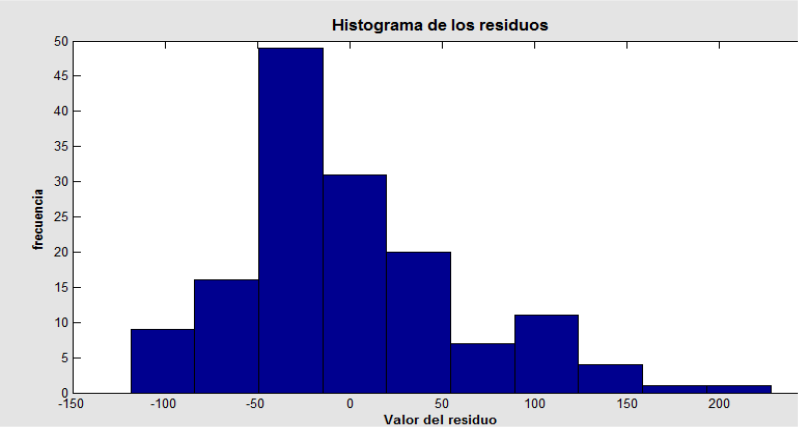

Figura 6.17: Histograma de residuos para la variable Y_2-Fuerza axial

En estos histogramas se puede ver cómo, para la fuerza de corte, el gráfico de la distribución del error es aproximadamente simétrico, es decir tiene la misma varianza para todo el rango de valores, y posee media cero, por lo que el modelo es bueno para la fuerza de corte. Sin embargo, en el caso de la fuerza axial esto no es así, observando una distribución asimétrica. Además en este último caso se observan mayores residuos.

Otro gráfico útil para saber la bondad del ajuste obtenido mediante el modelo lineal es el *Gráfico Y real-Y estimada*. En las siguientes figuras se muestran ambas curvas. En ellas se puede ver que el modelo aproxima peor las salidas más grandes. También vemos es que el modelo lineal aproxima medianamente bien los datos, mejor de lo esperado.

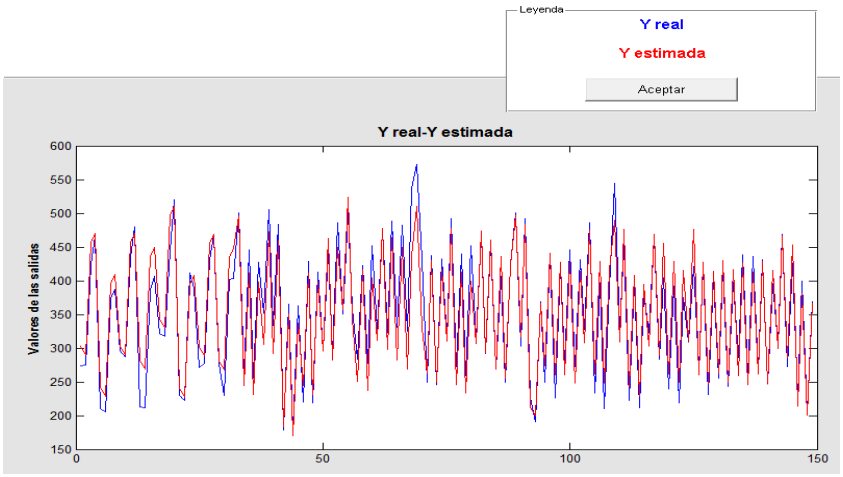

Figura 6.18: Gráfico Y real-Y estimada para la fuerza de corte

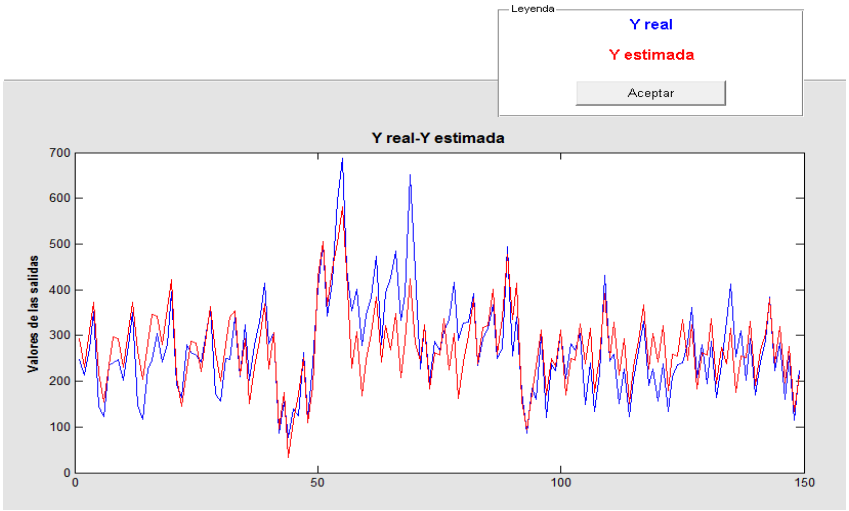

Figura 6.19: Gráfico Y real-Y estimada para la fuerza axial

Por último, otra herramienta útil de la cual dispone el programa es el gráfico de dispersión *Y estimada vs residuos*, que se muestran en las figuras 6.20 y 6.21. En estos gráficos, lo ideal sería que los residuos fuesen un ruido blanco, sin embargo en ambos casos podemos ver que esto no es así, por lo que el modelo no resulta muy adecuado. En el caso de la fuerza de corte, se observa una clara no linealidad en los residuos. Por otro lado, aunque el modelo no es adecuado, en este gráfico observamos unos residuos muy altos, podría tratarse de datos atípicos y sería mejor revisarlos. En el caso de la fuerza axial, se ve claramente que

los residuos tienen un comportamiento creciente, son mayores a medida que la fuerza axial es mayor. Igual que antes se observan algunos residuos muy altos con respecto al resto, podrían deberse a observaciones atípicas.

Para conocer qué observaciones corresponden a estos residuos tan altos, la GUI ofrece la opción *Identificar observaciones con residuos altos*. Por ejemplo, si para la Fuerza de Corte, se pide que nos indique qué observaciones tienen residuos más altos de 60, nos indica que son las observaciones que se muestran en la figura 6.21b.

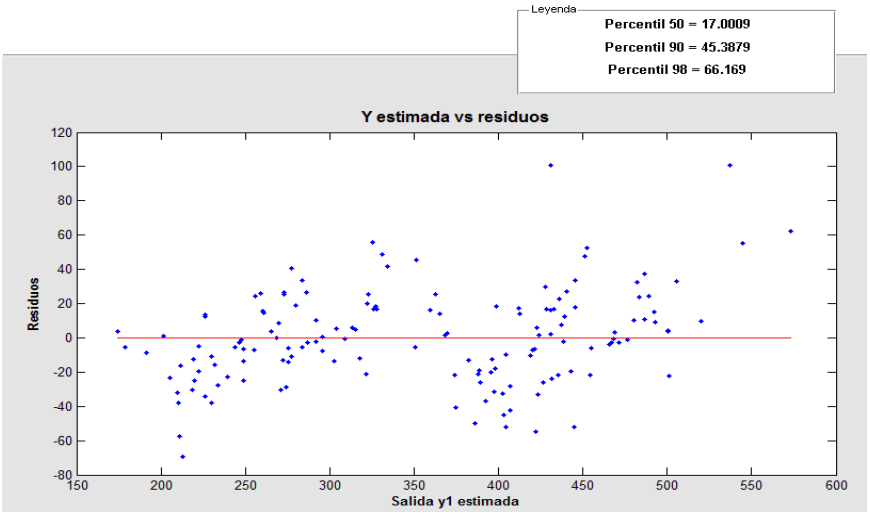

Figura 6.20: Gráfico Y estimada vs residuos para la fuerza de corte

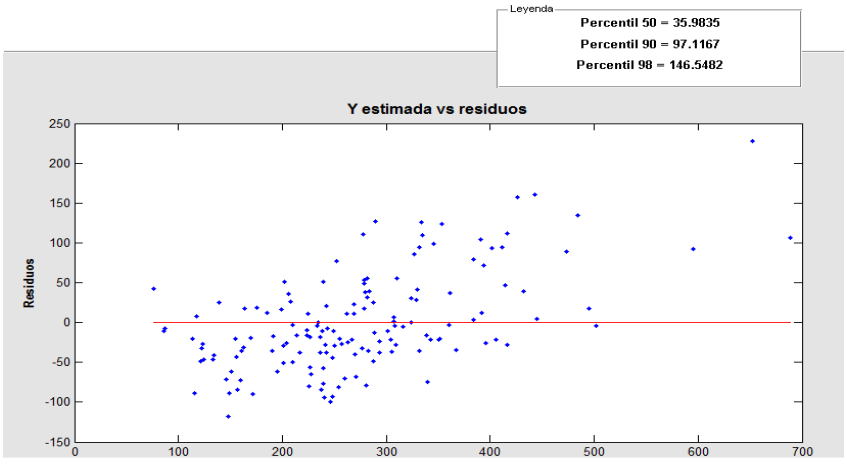

Figura 6.21: Gráfico Y estimada vs residuos para la fuerza axial

Posibles datos erróneos												
Posición	Dato		Salida		Residuo							
13	0.0200	0.8000	0.0000	0.0000	0.0873	1.3090	120.0000	0.0500	----	212.6000	----	-69.6149
68	0.0000	0.8000	0.0000	0.0000	0.0873	1.3090	240.0000	0.1000	----	537.2000	----	100.9485
69	0.0000	0.8000	0.0000	0.0000	0.0873	1.4835	120.0000	0.1000	----	573.0000	----	62.4359
70	0.0000	0.8000	0.0000	0.0000	0.0873	1.4835	240.0000	0.0500	----	430.8000	----	100.5762

Figura 6.21b: Posibles datos erróneos

6.1.3.-Arquitectura de red.

Como ya se comentó anteriormente, los histogramas de cada una de las variables muestran que las variables de entrada son discretas, por lo que se utilizara una red de base radial que se adecúa mejor a esta estructura de datos que las redes backpropagation, por su carácter local en la aproximación.

Para decidir mejor que parámetros fijar para la red, y puesto que en este tipo de redes el spread es un parámetro muy importante, se utilizará la opción que da el programa de realizar el entrenamiento de la red con varios spreads para comparar el comportamiento. Se escogerá un intervalo entre 0.5 y 8.5. Posteriormente se escogerá el mejor y se realizará un estudio individualizado.

Por otro lado se fijará un error objetivo de 10^{-4}, y un máximo número de neuronas de la tercera parte de datos que poseemos, es decir, 50. Por supuesto la red tendrá ocho entradas y dos salidas.

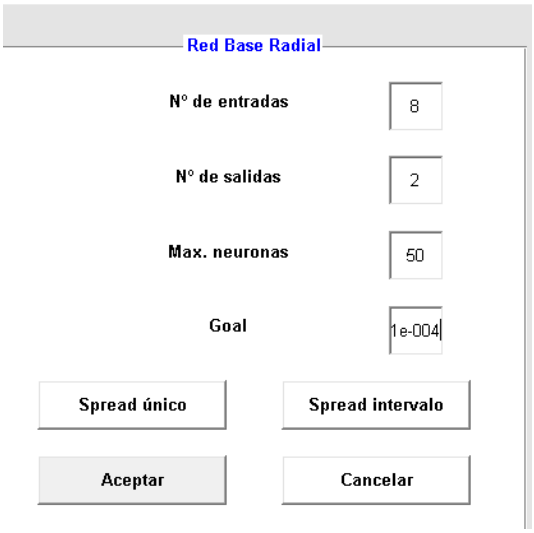

Figura 6.22: Arquitectura para definir la arquitectura de la red de base radial

Todo esto se fija en panel que se despliega al acceder a la herramienta *Base radial*. En las figuras 6.22 y 6.23 se muestran este panel con los datos arriba expuestos introducidos y

el panel que nos permite fijar el intervalo de valores para el spread. Una vez aceptado veremos en el panel de información del proceso los datos de la arquitectura de red escogida.

Figura 6.23: Panel para definir el intervalo del spread

6.1.4.-Parámetros de entrenamiento y entrenamiento.

Al utilizar redes de base radial, en el entrenamiento únicamente tendremos que definir el tratamiento que se realizará a las entradas, el tipo de entrenamiento que se utilizará y especificar el fichero de datos de entrenamiento.

En primer lugar especificaremos el tratamiento que se aplicará a las entradas. Se ha escogido escalar las entradas para limitar los valores que puede dar la red, y de esta manera mejorar el entrenamiento de la misma. Para ello se accede a la herramienta *Tratamiento de entradas*, y se escoge la opción escalada, tal y como se muestra en la figura 6.24.

Seguidamente decidimos el tipo de entrenamiento. Se escogerá entrenamiento cross validation porque tenemos únicamente un fichero de datos que además contiene muy pocos, y con este tipo de entrenamiento podemos visualizar la capacidad de generalización que tiene la red. En este tipo de entrenamientos hay que definir dos parámetros, el número de datos de generalización y el número de datos de cross validation. Se escogen diez datos de validación,

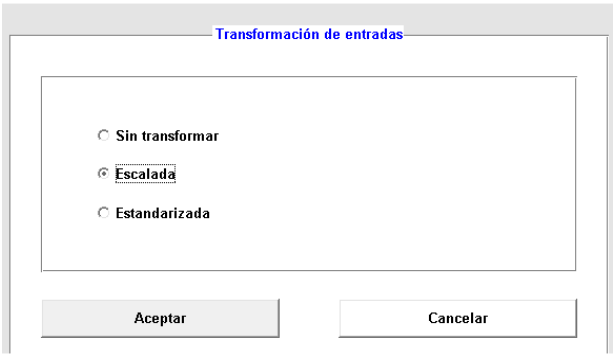

Figura 6.24: Tratamiento de entradas

por la sencilla razón de que únicamente se disponen de 150 datos de entrenamiento. Una vez

elegido este parámetro se decide el número de veces que se realiza cross validation. Se escogerán valores no muy grandes porque este primer entrenamiento es para decidir que spread escoger. Por esta razón se realizarán 30 iteraciones.

En la figura 6.25 se muestra el panel que se despliega al elegir la opción *Cross validation* con el valor de los parámetros introducidos.

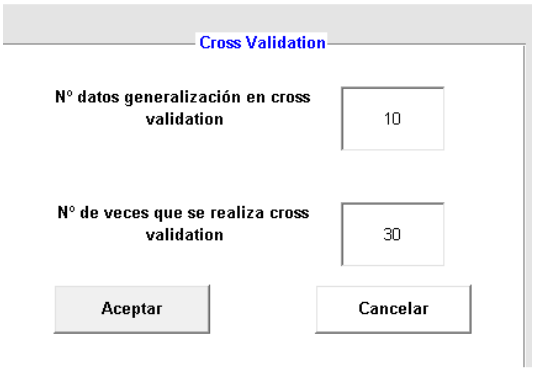

Figura 6.25: Panel para definir los parámetros de *Cross validation*

Finalmente se carga el fichero de entrenamiento de la misma forma que se hizo en el análisis previo de datos. Una vez realizado esto, se procede al entrenamiento de la red con distintos spreads. En la figura 6.26 se muestra los gráficos mostrados durante el entrenamiento. La ventana que aparece al final del entrenamiento no se muestra en este documento porque la información que muestra (el error medio entre todas las iteraciones) no es de gran utilidad.

Una vez acabado el entrenamiento, se estudia el error relativo medio que comete la red para cada spread y cada salida. Estos gráficos aparecen en las figuras 6.27 y 6.28.

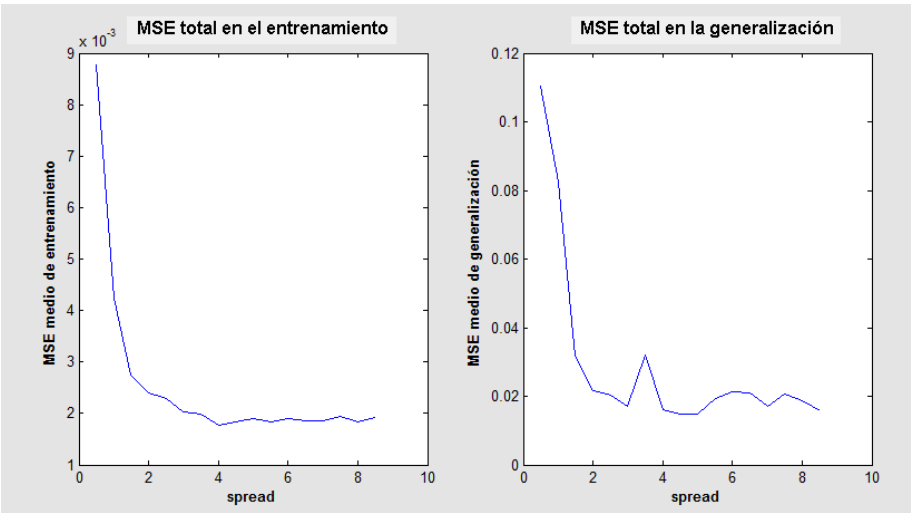

Figura 6.26: Gráficos de el error cuadrático medio vs spread

Para obtener los gráficos de la figura 6.27 y 6.28, únicamente hay que acudir a

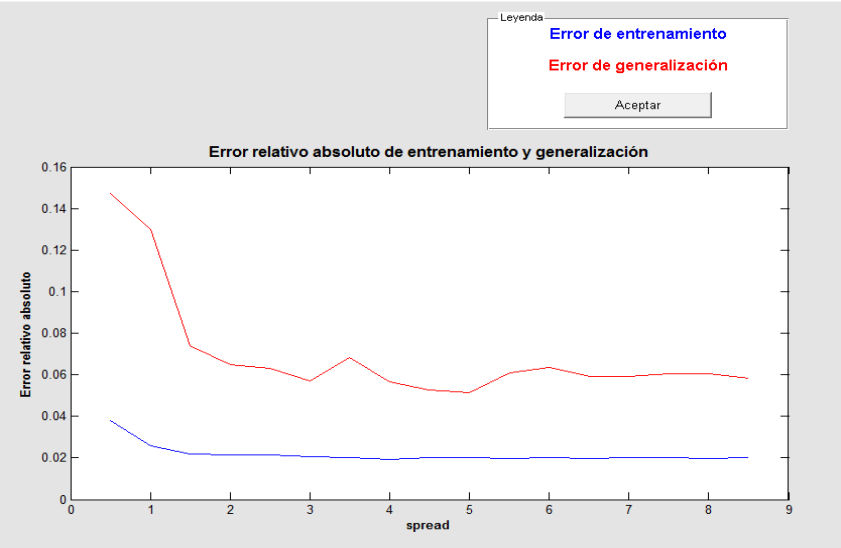

Figura 6.27: Gráfico del error relativo absoluto para la fuerza de corte

Postproceso, elegir la gráfica *Error relativo absoluto* y elegir la salida correspondiente, tal y como se indica en 5.3.6. En dichas gráficas observamos que para ambas salidas se obtiene un error relativo menor si se utiliza spread igual a cinco. Además observamos como la fuerza axial es peor aproximada que la fuerza de corte para cualquier valor de spread, estando sus errores por encima del 10%.

Una vez elegido el spread, se procede a realizar un entrenamiento individualizado. Para ello, en vez de elegir la opción *Spread intervalo* en la arquitectura de red de base radial, escogemos *Spread único*. Además y para obtener resultados más fiables se realizarán más iteraciones de cross validation, aumentándolas a 100. En la figura 6.29 se muestra la ventana que aparece al final del entrenamiento en la que esta vez sí, la información es

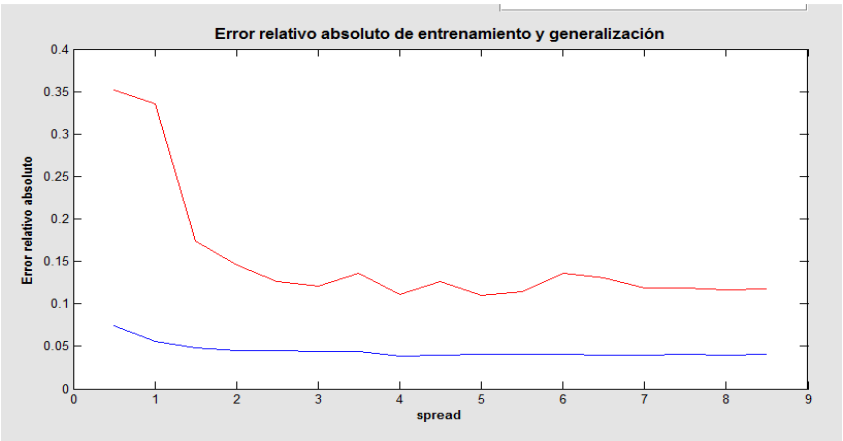

Figura 6.28: Gráfico del error relativo absoluto para la fuerza axial

relevante.

Figura 6.29: Ventana que aparece al final del entrenamiento

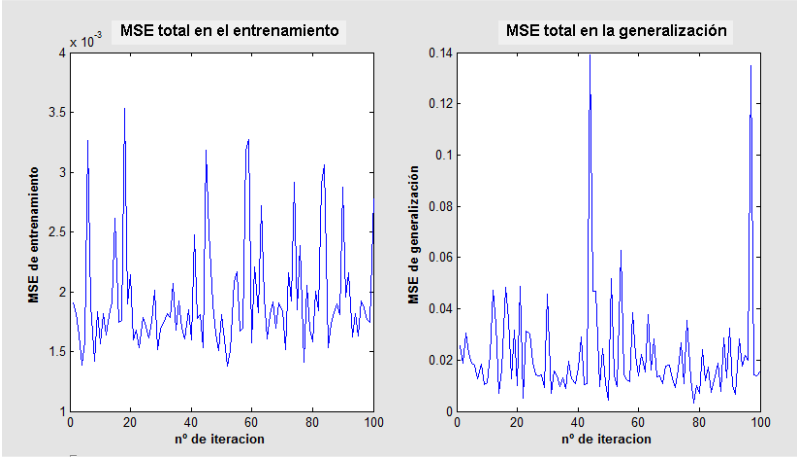

Figura 6.30: Errores cuadráticos medios

Vemos que los errores medios para ambas salidas no varían mucho, de hecho son casi iguales. En la figura 6.30 se muestran los errores cuadráticos medios para el entrenamiento y generalización. En ella vemos como estos errores varían considerablemente de una iteración a otra, lo que nos invita a pensar que hay determinado grupo de datos que son representativos de la muestra y otro que no. Para obtener resultados que no dependan de la "suerte" de la elección de datos se realiza cross validation. Para asegurarnos de que se ha realizado suficiente número de iteraciones se repite el entrenamiento con 200 y se comparan los errores medios, viendo que éstos apenas varían, lo que nos indica que el número de iteraciones está bien.

A continuación se va a utilizar la herramienta de *Postproceso* para estudiar la aptitud de la red.

6.1.5.-Postproceso.

Esta herramienta funciona tal cual se indica en el apartado 5.3.6. Únicamente hay que tener en cuenta que aparece un panel donde hay que indicar el tipo de gráfico que se desea y la salida.

En primer lugar vamos a estudiar el error relativo para cada una de las salidas. Vemos en las figuras 6.31 y 6.32 la evolución del error relativo en cada una de las iteraciones realizadas durante el cross validation. Como era de esperar, varían considerablemente de una iteración a otra pero vemos que los valores medios son semejantes a los obtenidos en el primer entrenamiento donde variábamos el spread. Vemos también que la fuerza axial se aproxima peor que la fuerza de corte. Además vemos como la red aproxima bastante mejor los datos de entrenamiento que los de generalización. Esto nos invita a pensar que han podido ocurrir dos cosas, o que se ha producido sobreaprendizaje o que no hay suficiente cantidad de datos de entrenamiento que sean representativos del problema. Posiblemente sea esto último

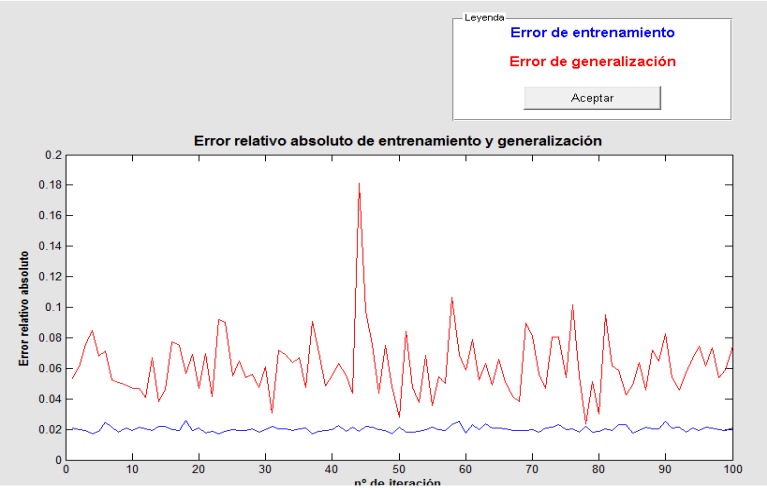

Figura 6.31: Evolución del error relativo absoluto para la fuerza de corte

debido a los pocos datos que poseemos y a la gran cantidad de variables que existen.

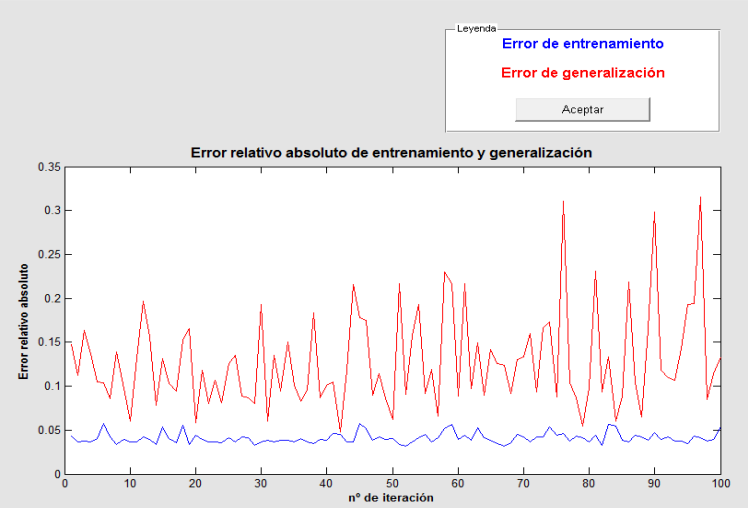

Figura 6.32: Evolución del error relativo absoluto para la fuerza axial

En las figuras 6.33 y 6.34 se muestran los gráficos que comparan las salidas estimadas por la red con las salidas exactas en la fase de entrenamiento y generalización respectivamente.

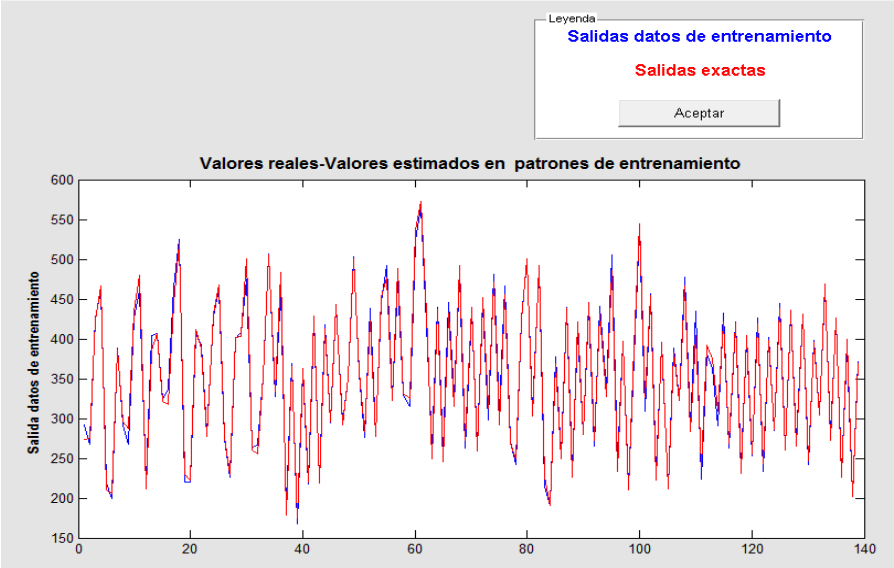

Figura 6.33a: Gráfico valores reales-valores estimados en fase de entrenamiento

En las figuras 6.33a y 6.33b vemos como la red aproxima muy bien los patrones de entrenamiento, sin embargo aproxima algo peor los datos de generalización, aunque la aproximación es aceptable, sobre todo para la fuerza de corte.

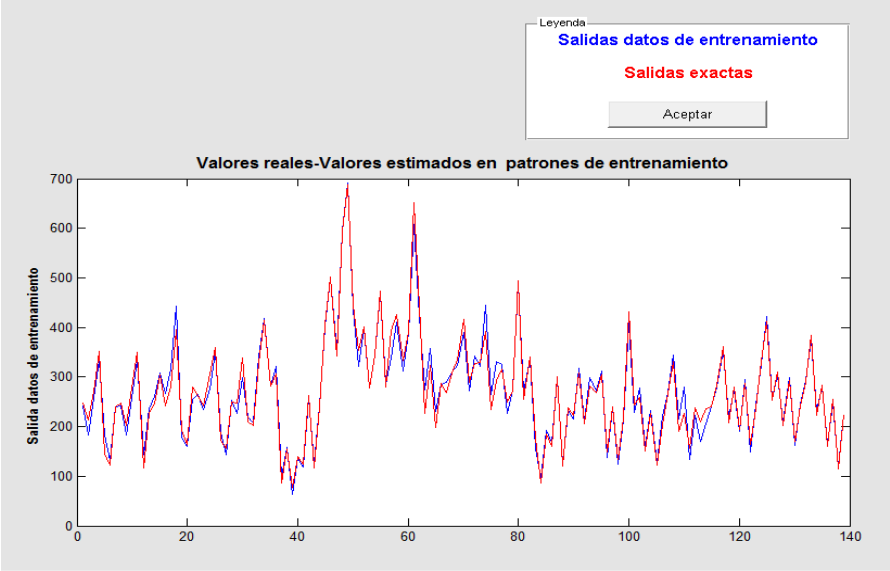

Figura 6.33b: Gráfico valores reales valores estimados en fase de entrenamiento para fuerza axial

Para analizar la bondad de ajuste del modelo se analizan los gráficos de dispersión de los residuos vs las variables de salida, en la fase de entrenamiento, que se representan en la figura 6.36a y 6.36b. En estos gráficos podemos ver un comportamiento más aleatorio de los

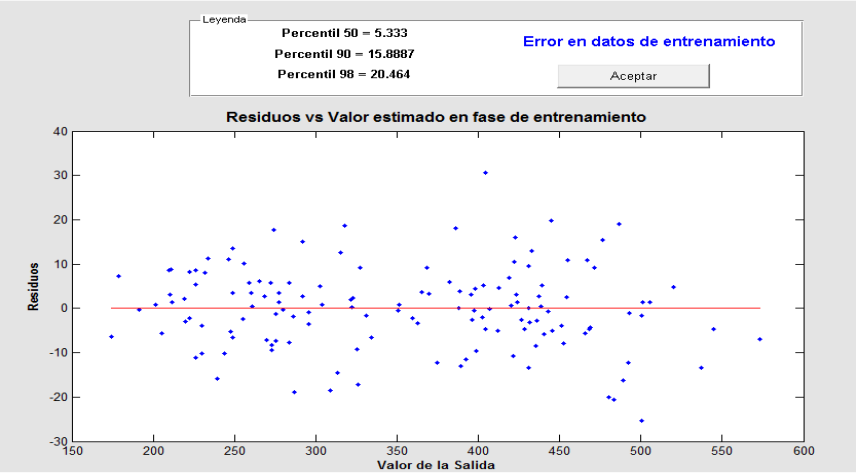

Figura 6.36a: Gráfico de dispersión Residuos-Valores estimados en fase de entrenamiento para la fuerza de corte

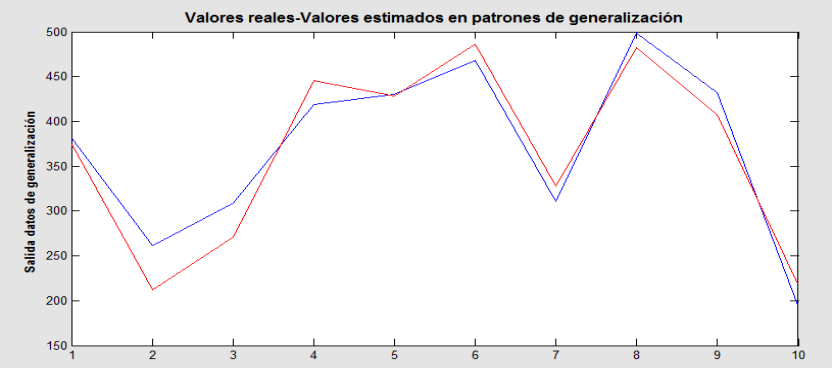

Figura 6.34a: Gráfico valores reales valores estimados en generalización para la fuerza de corte.

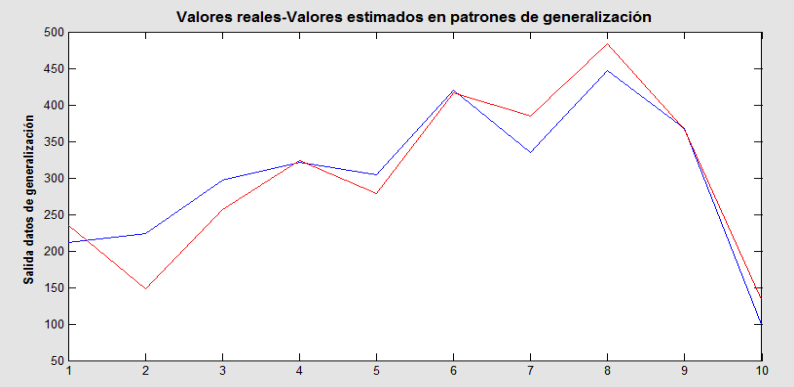

Figura 6.34b: Gráfico valores reales valores estimados en generalización para la fuerza axial.

residuos que en el caso del modelo lineal.

Aún así la red debería de mejorarse para que fuera apropiada para resolver el problema. Una posible mejora sería obtener más datos que permitieran a la red tener más información sobre el problema. Cabe destacar que se observa que donde peor se comporta la red es para fuerzas menores, posiblemente, y como se vio en los histogramas, porque disponemos de menor cantidad de datos en esta región.

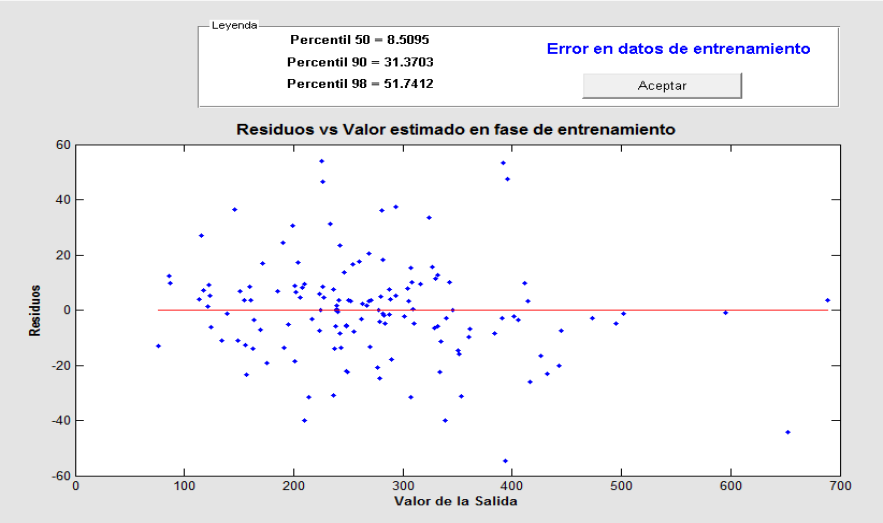

Figura 6.36b: Gráfico de dispersión Residuos-Valores estimados en fase de entrenamiento para la fuerza axial

Una vez visto los resultados y decidido que esta red es apropiada para el problema, procedemos a guardar los resultados. Finalmente podemos realizar un entrenamiento simple utilizando todos los datos del fichero de entrenamiento.

6.1.6.-Guardar resultados de entrenamiento.

Para ello se procede de la misma forma que se indica en el apartado 5.3.8. En este caso, al fichero se le dará el nombre de *FuerzasTaladrado* (figura 6.37). En la figura 6.38 vemos como el programa genera un documento de texto en el directorio de trabajo.

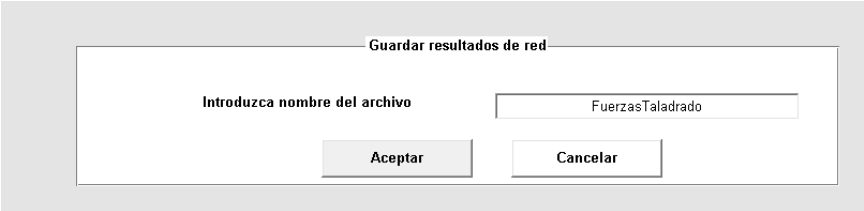

Figura 6.37: Guardar resultados

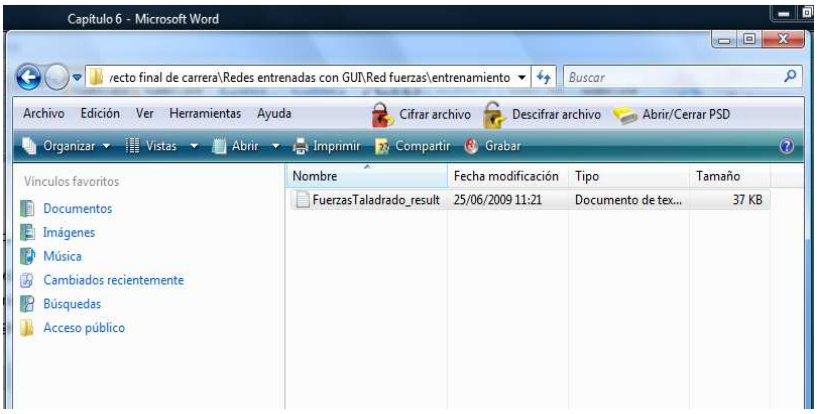

Figura 6.38: Fichero de resultados de entrenamiento

6.2.-Diseño de una red neuronal para obtener la curva de potencia de un generador eólico.

6.2.1.-Introducción.

En este ejemplo se intenta conseguir una red neuronal que estime la curva de potencia frente a la velocidad del viento de un generador eólico. Para ello se disponen de 1200 datos de la velocidad y orientación del viento y la potencia producida por el generador eólico. Se intentará conseguir una red que modele bien el problema, para posteriormente simularla con distintas velocidades de viento con el fin de obtener la curva de potencia real del aerogenerador.

Para ello se estudiarán diversos modelos con el fin de conseguir un modelo apropiado. Se probará inicialmente una red con dos entradas (orientación y velocidad), donde, entre otras cosas, se estudiará la relevancia en el modelo de la orientación del viento, y finalmente se estudiará una red de dos entradas, donde las entradas serán velocidad y velocidad al cuadrado. Se compararán los dos modelos y se decidirá cuál es mejor. Finalmente se concluirá si el empleo de las redes neuronales es apropiado en este caso.

6.2.2.-Análisis previo de datos.

Igual que se hizo en el caso anterior se estudiará las propiedades de la muestra de datos de la que se dispone. En esta sección únicamente se mostrarán los resultados más importantes.

En primer lugar vamos a estudiar la distribución de cada una de las variables mediante la obtención de los histogramas, los cuales se muestran en las figuras 6.39, 6.40 y 6.41.

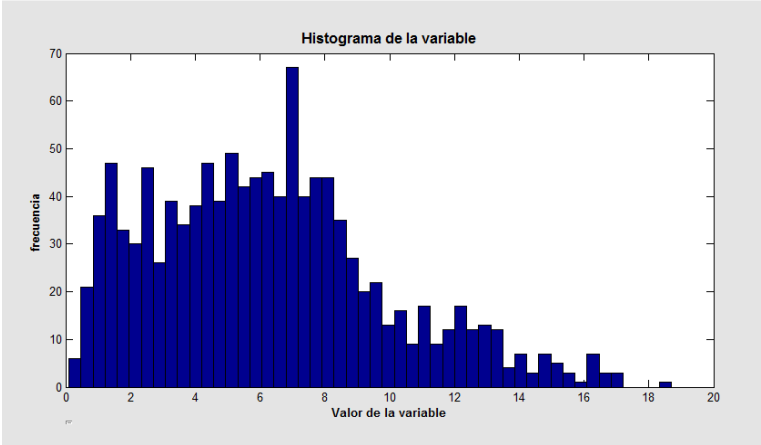

Figura 6.39: Histograma para la variable velocidad de viento

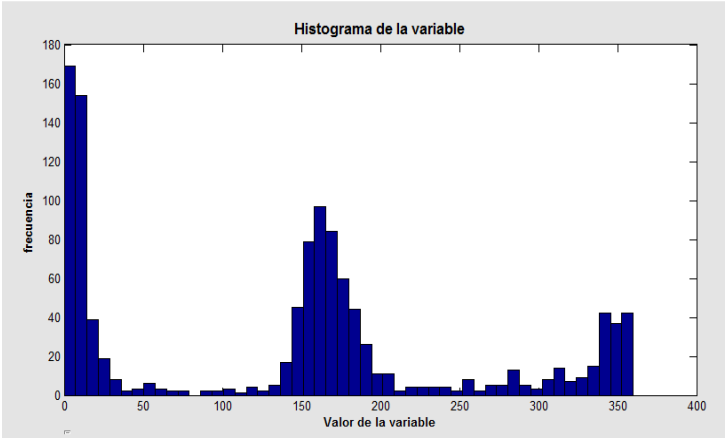

Figura 6.40: Histograma para la variable orientación del viento

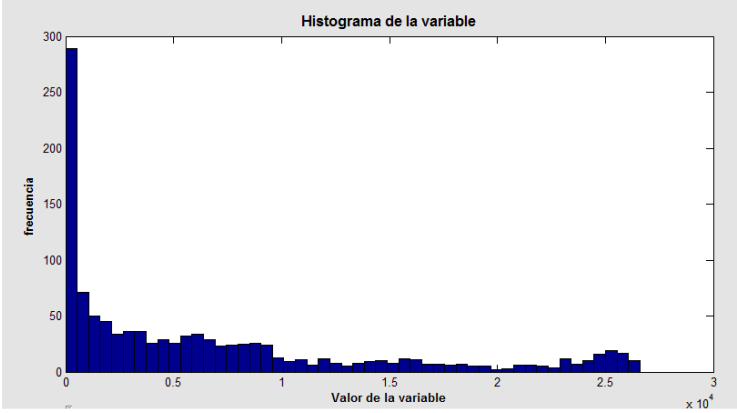

Figura 6.41: Histograma de la variable de salida potencia generada

Observamos en primer lugar que existen bastantes datos. En segundo lugar, existen datos a lo largo de todo el rango de velocidades de viento, quedando algunos huecos para velocidades elevadas. En cuanto a la orientación del viento, en la figura 6.40 se observa que la mayoría de las observaciones se concentran en torno a 0 ó 360º y 180º. Ambas son orientaciones de viento que dan como resultado potencias parecidas, por lo que se estudiará un modelo que no considere la orientación del viento. Por último, en la variable de salida observamos que hay una concentración de datos para potencias pequeñas existiendo un pico para potencia igual a cero.

A continuación se mostrarán los gráficos de cada variable de entrada con las variables de salida para ver la correlación que existe. En la figura 6.42 se muestra un diagrama de dispersión donde en el eje x se representa la velocidad del viento y en el eje y la potencia

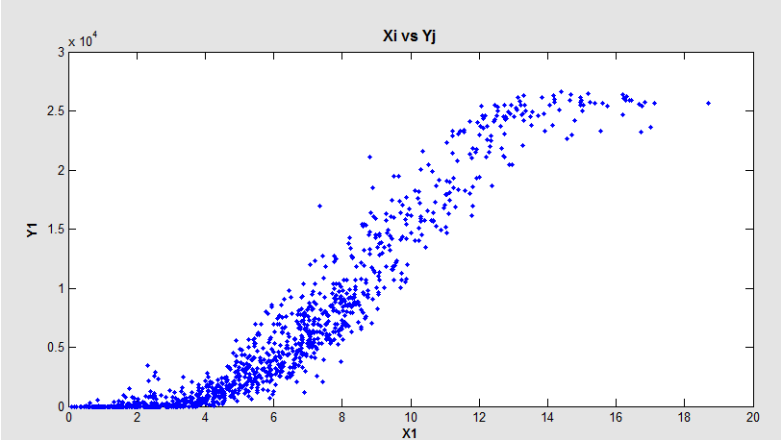

Figura 6.42: Diagrama de dispersión de potencia vs velocidad del viento

generada. Se observa en los extremos dos regiones; una para velocidades pequeñas donde el generador no genera nada, y otra para velocidades de viento alta donde el generador produce una potencia eléctrica máxima que no varía a pesar del aumento de velocidad del viento. En la región central se observa una dependencia no lineal de una variable frente a otra, además

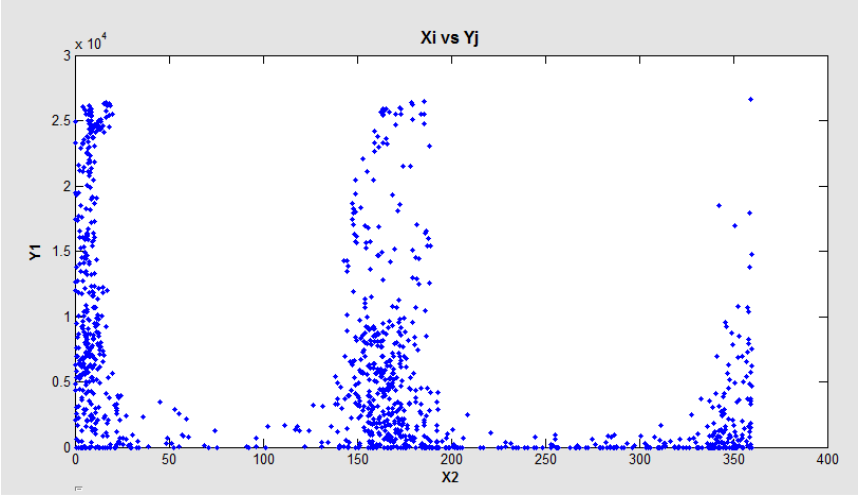

Figura 6.43: Diagrama de dispersión de potencia vs orientación del viento

de una gran dispersión de datos para una misma velocidad, posiblemente provocada por la dependencia de la orientación o por otras variables que en principio en el modelo no se tienen en cuenta.

En la figura 6.43 se muestra el mismo gráfico que en el caso anterior pero esta vez se ha utilizado orientación del viento en vez de velocidad. Se observa que para las orientaciones en torno a 0, 180 y 360º no parece que exista correlación entre las variables. Únicamente cuando esta fuera de estos valores la orientación afecta considerablemente a la potencia ya que la disminuye drásticamente. Es una de las principales razones por la que se estudiará un modelo que no tenga en cuenta la orientación para ver si se comporta mejor que teniéndolo en cuenta.

Seguidamente y al igual que en el caso anterior vamos a obtener un modelo lineal para el problema utilizando regresión lineal múltiple, con el objeto de detectar posibles datos erróneos y obtener una referencia que nos permita evaluar las bondades de la red. Para ello se volverá a utilizar las herramientas del menú *Modelo de regresión lineal*. A continuación se mostrarán los resultados más importantes.

En primer lugar en la figura 6.44 se muestra el histograma de los errores que comete el modelo lineal ante los datos que poseemos. Observamos que no existe simetría en la distribución, y que los errores siguen una clara no linealidad, por lo que el modelo lineal no es bueno. Además aparecen dos datos con residuos un poco más altos que el resto. Intentaremos con el resto de herramientas discernir si son datos erróneos o no.

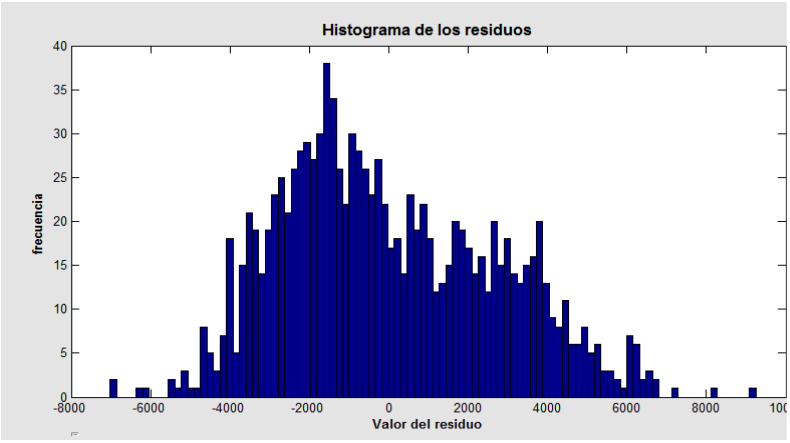

Figura 6.44: Histograma de residuos del modelo lineal

En el gráfico Y real vs Y estimada por el modelo no podemos distinguir estos errores. Pero si observamos que la mayoría de los errores más altos que comete el modelo son con potencias cero y con las potencias elevadas, ya que en estas regiones, y como se observa en la figura 6.42 existe un cambio de comportamiento en el modelo real que el modelo lineal no aproxima bien.

En el diagrama de dispersión de los residuos de la figura 6.46 sí observamos en la parte superior estos errores, pero no parece sobresalir excesivamente del resto, por lo que no se buscarán datos erróneos.

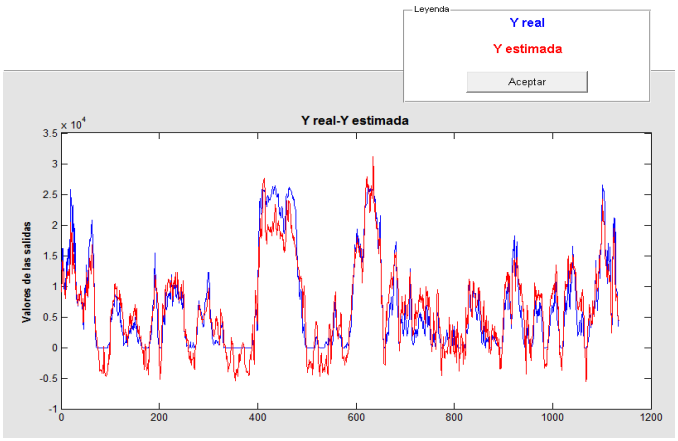

Figura 6.45: Gráfico Y real vs Y estimada

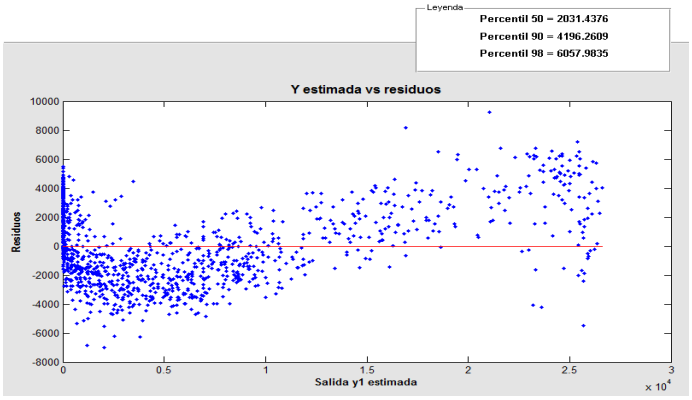

Figura 6.46: Gráfico de dispersión de residuos

En el gráfico de dispersión también se muestra la información de que el 90% de los errores están por debajo de 4196W. Este valor nos servirá como referencia para ver si estamos haciendo las cosas bien cuando entrenemos las redes neuronales.

6.2.3.-Elección de arquitectura de red.

Vista la distribución continua de los datos que poseemos, y también para mostrar un ejemplo de este tipo de redes, se utilizará una red backpropagation. Según la literatura consultada sobre redes neuronales, el número de capas ocultas no influye significativamente en el comportamiento de la red, por ello se utilizará una única capa oculta. Para un primer modelo, en el que se considera como entradas la velocidad y orientación, se estudiará el comportamiento de la red con 5, 10 y 20 neuronas en la capa oculta para ver cómo cambia. Asimismo, y puesto que se van a escalar las entradas, se utilizará en la capa oculta la función

de activación *tansig* y en la capa de salida *purelin* para que la red tenga posibilidad de dar salidas negativas.

6.2.4.-Parámetros de entrenamiento.

La función de error que se va a utilizar en el algoritmo de optimización de la red es la que tiene el programa por defecto (error cuadrático medio) puesto que las salidas están escaladas y presumiblemente los pesos no serán muy altos. Como ya se ha comentado, se escalarán las entradas porque de este modo se mejora el entrenamiento de la red. Para estudiar el comportamiento de las diferentes redes que se van a modelar se empleará cross validation con 50 datos de validación y generalización y cien iteraciones en cada modelo. Por último el algoritmo de optimización empleado va a ser el de Levenberg, donde todos los parámetros son los que tiene por defecto el programa salvo la tasa de aprendizaje que se ha fijado en 0.5. Todos estos parámetros se definen en el programa tal y como se explica en el apartado 5.3.5.

6.2.5.-Entrenamiento de los diversos modelos.

Como se ha comentado anteriormente, primero se decidirá el número de neuronas de la capa oculta. Para ello, se estudiarán el comportamiento de tres redes con 5, 10 y 20 neuronas. Todas tendrán dos entradas (velocidad y orientación del viento) y una entrada (potencia eléctrica). A continuación se muestran los resultados. En la figura 6.47 se muestran el error cuadrático medio de entrenamiento y generalización para la red de 5 neuronas en la capa oculta, en la 6.48 los resultados para la de 10 neuronas y en la 6.49 los resultados para

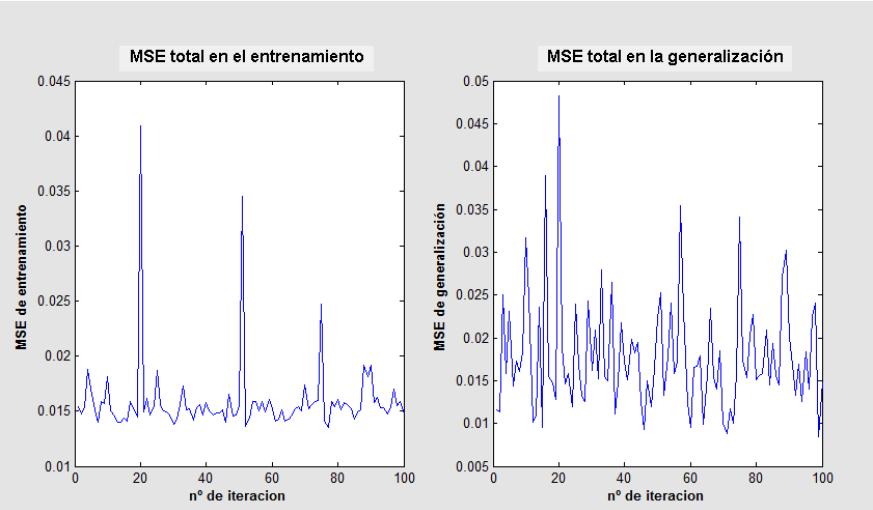

Figura 6.47: Error cuadrático medio para la red de cinco neuronas

la de 20 neuronas. En la tabla 6.1 se muestran los valores medios.

	MSE de entrenamiento	MSE de generalización
5 neuronas	0.126	0.134
10 neuronas	0.123	0.132
20 neuronas	0.125	0.131

Tabla 6.1: Errores cuadráticos medios

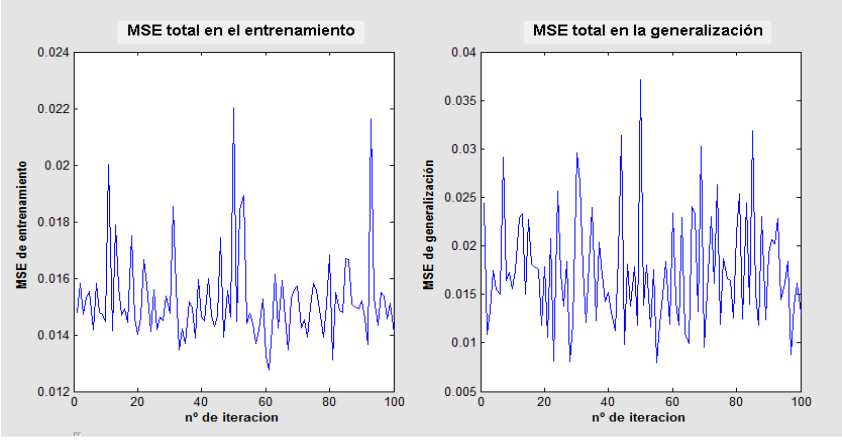

Figura 6.48: Error cuadrático medio para la red de diez neuronas

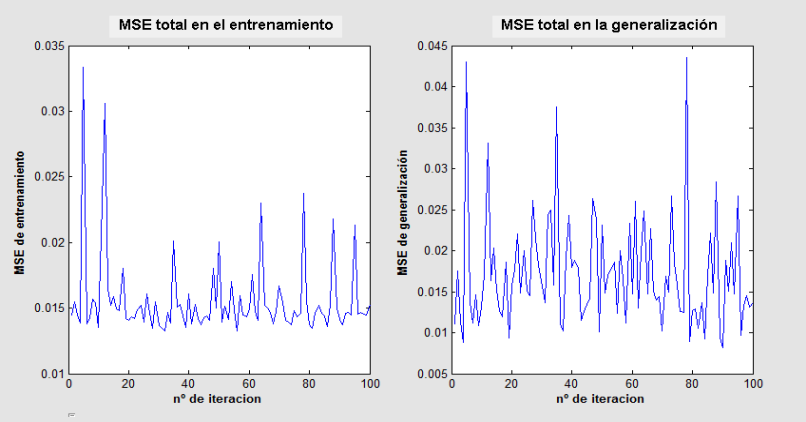

Figura 6.49: Error cuadrático medio para la red de veinte neuronas

En los gráficos se observa como existe cierta variación de los errores cuadráticos medios en cada iteración. Esto muestra que la aptitud del entrenamiento va a depender de los datos que se escojan. Por eso mismo se emplea cross validation, para obtener parámetros de comportamiento independiente de la división de datos que se haya realizado, como los

mostrados en la tabla 6.1. En ellos se infieren dos cosas. Por un lado se observa que el número de neuronas en la capa oculta no influye significativamente. Por otro lado, se observa que la generalización es tan buena como el entrenamiento.

Visto los resultados, para el resto de análisis que se llevarán a cabo en este caso, se empleará 10 neuronas en la capa oculta. Una vez decidida la arquitectura de la red, se estudiarán dos posibilidades, uno el expuesto anteriormente, otro, empleando nuevamente dos entradas, pero esta vez son la velocidad y la velocidad al cuadrado, por depender la energía eólica del cuadrado de la velocidad.

1º modelo: Velocidad y orientación.

Los gráficos del error cuadrático medio ya se muestra en la figura 6.48. A continuación se estudia como generaliza la red, y los errores que comete. Finalmente se simulará con distintas velocidades de viento y orientación para obtener las curvas de potencia, y compararlas con el diagrama de dispersión (esto último se realizará con herramientas de Matlab).

En la figura 6.50 se muestra el gráfico que compara los valores reales con los estimados por la red en los datos de generalización. En él se observa que la red aproxima mejor los datos que con el modelo lineal, pero existen ciertos datos en el que las diferencias son grandes todavía.

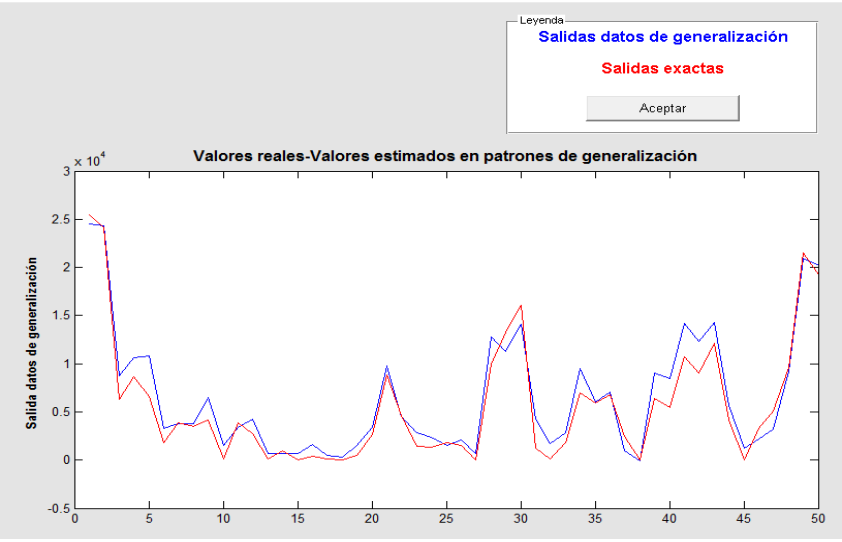

Figura 6.50: Gráfico Y real –Y estimada por la red en datos de generalización

En la figura 6.51 se muestra el diagrama de dispersión de los residuos frente a las salidas estimadas en la fase de entrenamiento. En él se observa que la distribución de errores presenta una ligera heterocedasticidad, los residuos tienen mayor dispersión para valores de

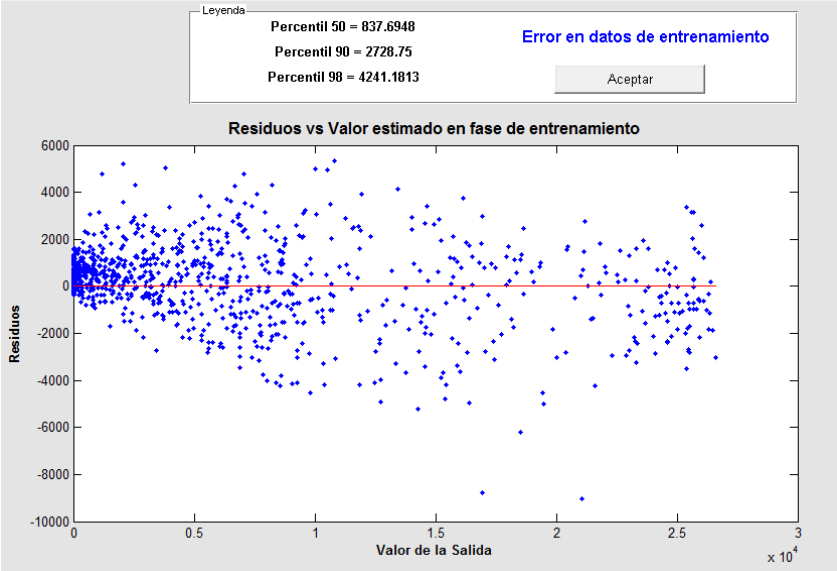

Figura 6.51: Diagrama de dispersión de residuos en fase de entrenamiento

potencia comprendidos entre 0.7 y 2. No se observa otro comportamiento anómalo.

Por último en la figura 6.52 se muestra el diagrama de dispersión de residuos pero para los datos de generalización. En él observamos que el 90% de los errores en valor absoluto están por debajo de 2900W. Se observa cierta mejora respecto al modelo lineal pero no parece suficiente. Aún así se seguirá con el estudio.

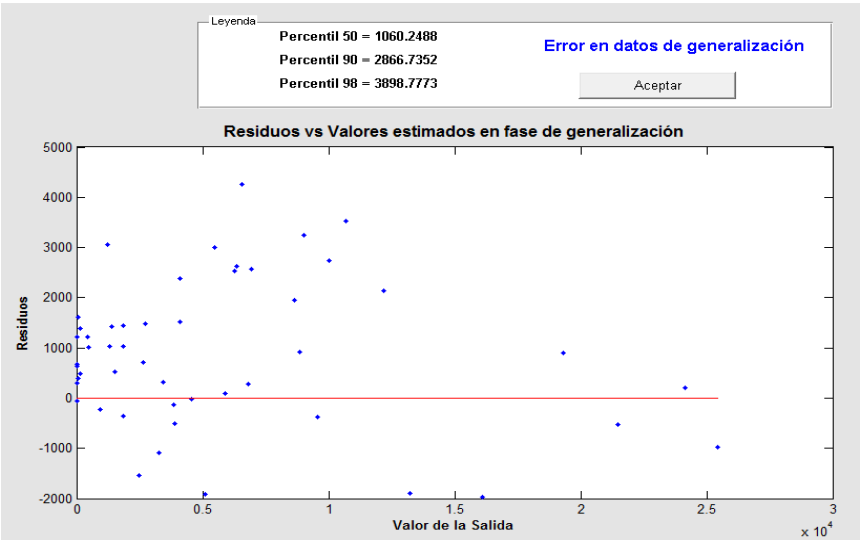

Figura 6.52: Diagrama de dispersión de los residuos en fase de generalización

Una vez obtenido los parámetros de comportamiento, y como no se observa apreciables diferencias entre los errores de entrenamiento y de generalización, para obtener la curva de potencia se utilizará una red con la misma arquitectura, pero que se entrenará con todos los datos (si se utilizarán datos de validación).

Posteriormente se simulará tres archivos de datos, los cuales en cada uno habrá valores de velocidades en todo el rango y cada 0.1m/s y la misma orientación, siendo para el primer archivo 0º, para el segundo 50º y para el tercero 180º. Se simularan utilizando la herramienta *Ejecutar la red con un nuevo conjunto de datos*. Esta herramienta generará los archivos de resultados para cada una de las simulaciones. Dichos archivos se utilizarán para obtener las curvas de potencia utilizando herramientas de Matlab.

En primer lugar, la herramienta solicita el nombre del fichero donde se encuentran los datos que se quieren simular y el nombre del fichero donde se guardarán los resultados. Dichos ficheros se guardarán en la carpeta de simulaciones dentro del directorio de trabajo.

En la figura 6.53 se muestra un diagrama de dispersión de los datos que poseemos superpuestos con las tres curvas de potencia calculadas, una para cada orientación. En ella la curva azul representa la curva de potencia para 0º, la curva roja para 50º, y la curva amarilla para 180º. Es destacable que estas curvas son muy parecidas, mostrando lo que se sospechaba en un principio, que la orientación no es una variable que sea relevante en el modelo. También destaca la gran dispersión en los datos y como las curvas de potencia estimadas por la red intentan minimizar la distancia a todos los puntos pasando por el "medio" de ellos. Este hecho nos invita a pensar que en el problema faltan variables de entrada de relevancia para que la red pueda aproximar mejor todos los datos. Quizás una variable relevante sea la temperatura.

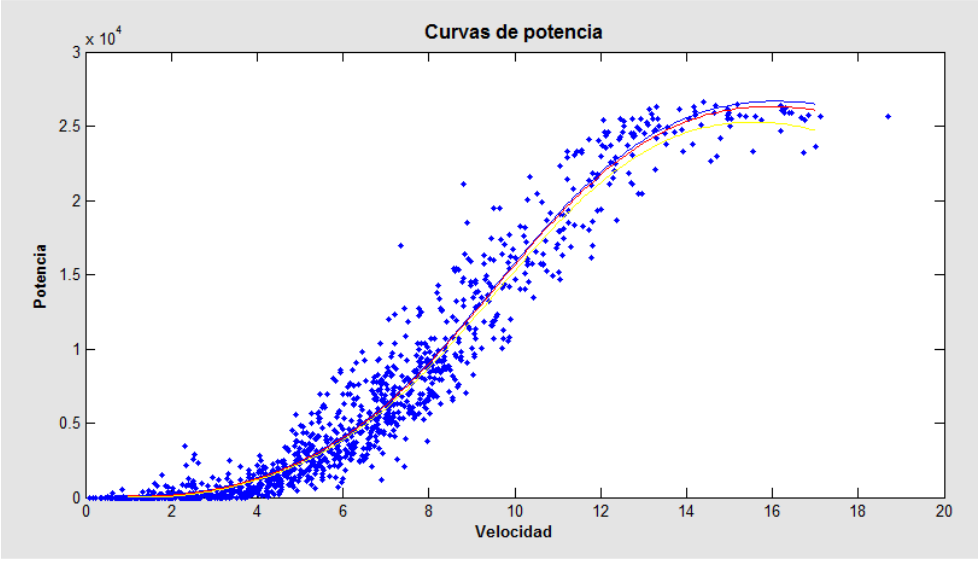

Figura 6.53: Representación de las curvas de potencia estimadas por la red frente a los datos de los que se dispone

En el segundo modelo a estudiar no se considera la orientación como variable de entrada a la red por la poca relevancia vista en el caso anterior que tiene ésta.

2º modelo: Velocidad y velocidad al cuadrado.

En este modelo se elimina como variable de entrada la orientación pero se añade el cuadrado de la velocidad de viento para intentar obtener mejoras en la red. Se emplearán la misma arquitectura, el mismo algoritmo de optimización y los mismos parámetros de entrenamiento que en el caso anterior, obteniendo los siguientes resultados.

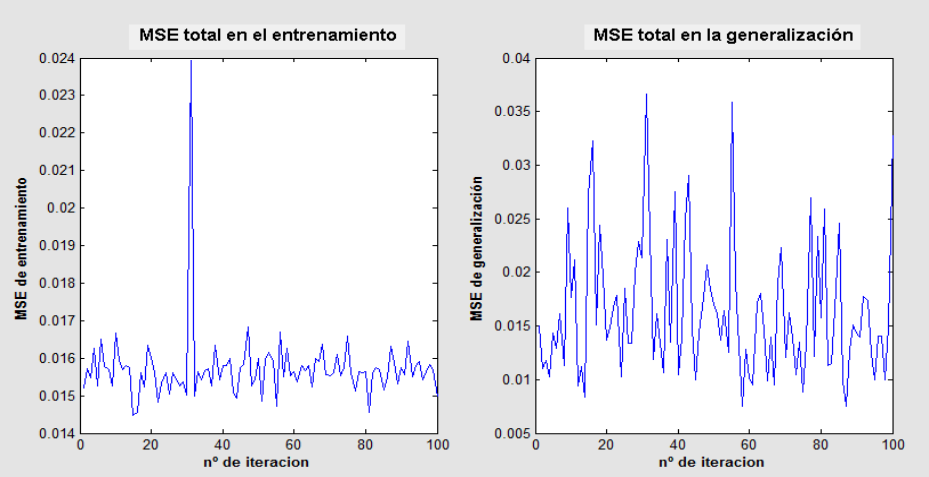

Figura 6.54: Error cuadrático medio de entrenamiento y generalización.

Los resultados son parecidos al caso anterior, únicamente se ha mejorado ligeramente los errores cuadráticos medios, que han pasado a 0.125 y 0.128 respectivamente. En la figura 6.55 se muestra como aproxima la red los patrones de generalización en comparación con los valores reales. Se aprecia cierta mejoría respecto al modelo anterior.

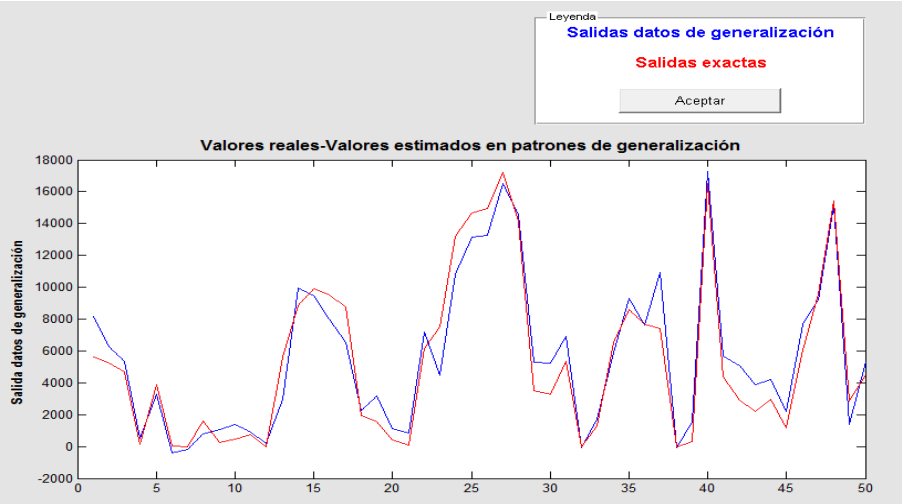

Figura 6.55: Valores reales-Valores estimados en generalización

En la figura 6.56 se muestra el diagrama de dispersión de errores en la fase de entrenamiento donde todavía se observa la misma heterocedasticidad que antes.

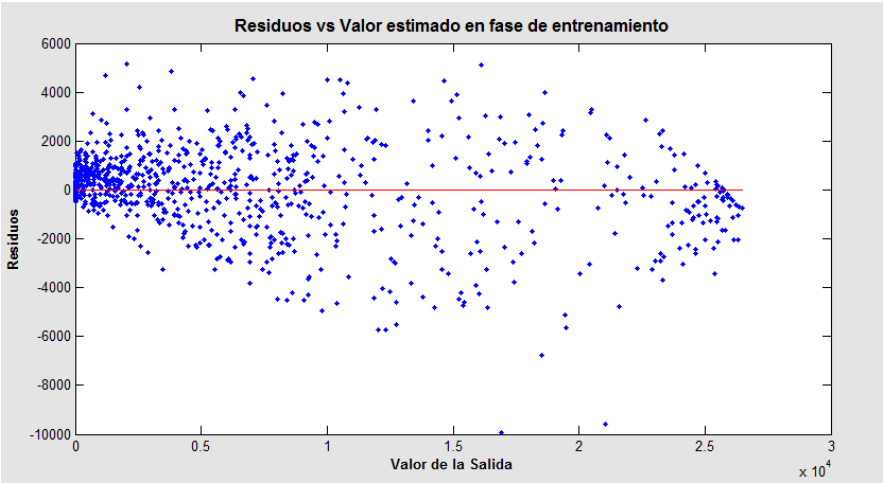

Figura 6.56: Diagrama de dispersión de residuos en la fase de entrenamiento

En la figura 6.57 se vuelve a representar un diagrama de dispersión de residuos pero esta vez en la fase de generalización. Se observa como ahora los errores son menores que en el modelo anterior, estando el 90% de ellos por debajo de 2333W y el 98% de 3270W.

De igual modo que se hizo con el caso anterior, esta red se simulará con diferentes datos de viento para obtener la curva de potencia, la cual se representa en la figura 6.58. Anteriormente se ha entrenado la red con todos los datos disponibles mediante la opción de *entrenamiento simple*.

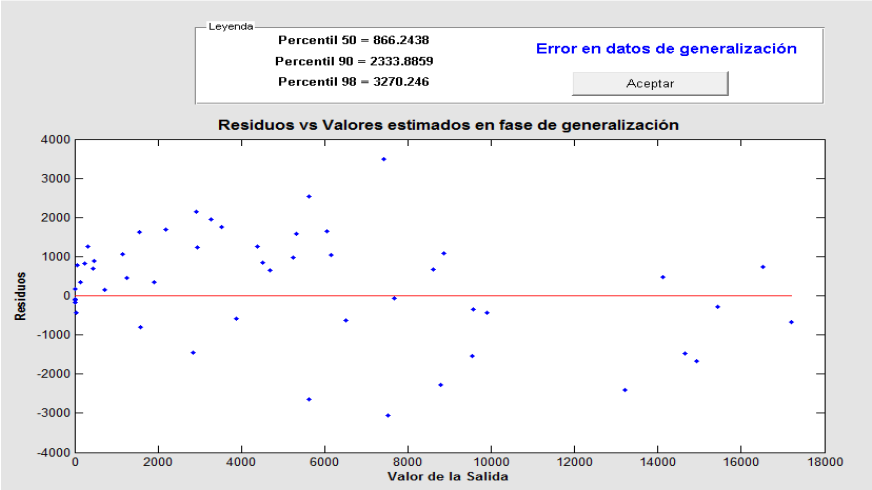

Figura 6.57: Diagrama de dispersión de los residuos en la fase de generalización

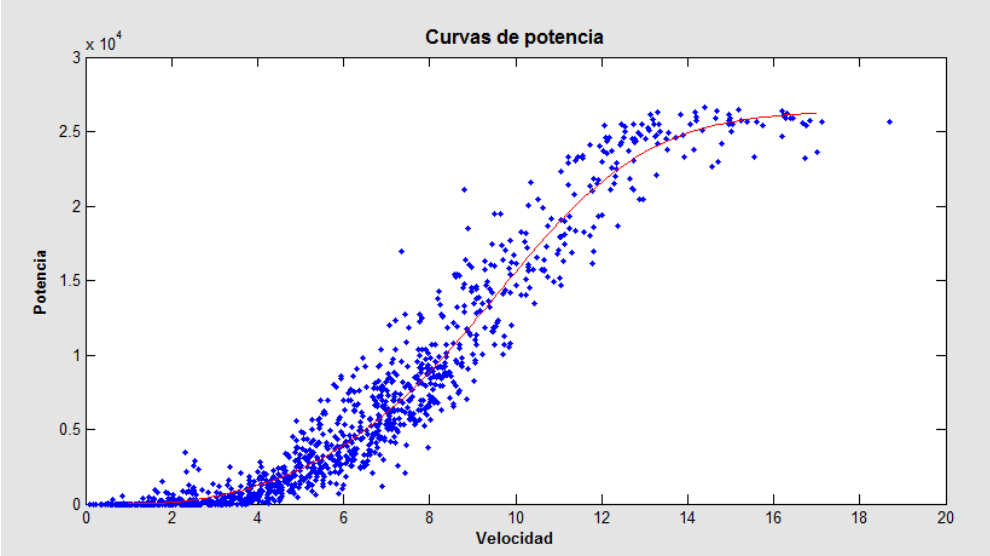

Figura 6.58: Curva de potencia estimada por el segundo modelo.

Básicamente la curva de potencia es similar a la estimada por el primer modelo, pero en los resultados visualizados durante el postproceso se aprecia una ligera mejoría en los mismos. Por lo tanto ésta será la red escogida. Con la herramienta *Guardar resultados de entrenamiento* obtenemos un fichero con un resumen de resultados, donde entre otras cosas muestra información de la arquitectura de red, parámetros de entrenamiento y salidas estimadas por la red ante los diferentes datos de entrenamiento.

6.2.6.-Conclusiones.

En el estudio hecho de este caso se observa como la curva estimada por las dos redes entrenadas pasa por el "medio" del diagrama de dispersión, y como la orientación del viento no influía significativamente. Además para una misma velocidad se observa una gran dispersión de resultados que no se pueden explicar únicamente con la orientación del viento. Por lo tanto, teniendo en cuenta esto, y la vista de los resultados, se puede llegar a la conclusión que en el problema faltan variables de entrada que justifiquen dicha dispersión, y que utilizando sólo la velocidad y orientación no se conseguirán aproximaciones mejores con ninguna otra red, porque la limitación no viene impuesta por la red, sino por como se ha modelado el problema al principio.

Capítulo 7: Conclusiones y futuros trabajos

En este proyecto se ha desarrollado una aplicación que, como se ha visto, es sencilla de utilizar pero que a la vez dispone de las herramientas necesarias para obtener un modelo de redes neuronales adecuado al problema que se analiza. Esto no signifca, que siempre se conseguirán resultados satisfactorios, en lo que a grado de precisión se refiere. Puede darse el caso de que los modelos de redes neuronales no sean los más apropiados para el problema analizado. Los ejemplos incluidos en el capítulo 6 permiten corroborar la sencillez y utilidad de la herramienta desarrollada.

Por ello, el autor estima oportuno concluir que se ha conseguido una herramienta que reúne gran parte de las características buscadas y que cumple con el principal objetivo del proyecto. Se ha conseguido mejorar la aplicación existente en Matlab, otorgando al usuario mayor flexibilidad y proporcionándole una herramienta más específica.

Dicha herramienta está abierta a posibles mejoras, algunas de las cuales podrían ser las siguientes:

- Introducción de nuevos algoritmos de entrenamiento, así como aumentar la flexibilidad de los ya existentes, es decir, que el usuario sea capaz de elegir mayor número de parámetros.
- Introducción de nuevos tipos de redes, pues si bien es verdad que las más utilizadas son las redes backpropagation y de base radial, en algunos tipos de problemas o aplicaciones podría ser más óptimo la utilización de otro tipo de red.
- Mejorar la visualización de resultados, añadiendo más opciones a los gráficos existentes y añadiendo otros que sean útiles para la evaluación del comportamiento de la red. Una posible opción sería que la aplicación sea capaz de graficar una curva estimada por la red (como se hizo en el segundo caso del capítulo 6, lo cual se hizo con ayuda de herramientas de Matlab).
- Mejorar las herramientas que permiten realizar un estudio previo de los datos, tales como incluir opciones que permitan estudiar el grado de correlación de las entradas con las salidas, y de esta manera ayudar al usuario a decidir que variables son relevantes en el problema.
- Por último, integrar esta aplicación con otras aplicaciones, como por ejemplo una que se está desarrollando actualmente, en las cual se trabaja con otra técnica no paramétrica denominada regresión local polinómica. Esta técnica puede servir como alternativa al empleo de redes neuronales en la modelización de un problema.

Capítulo 8: Bibliografía.

[1] Esqueda Elizondo, J.J.(2002).”Matlab e Interfaces Gráficas”. Documento de internet.

[2] Highman, D.J. (2000). “Matlab Guide”, Siam.

[3] Isasi Viñuela, P. (2004). “Redes Neuronales Artificiales: Un Enfoque Práctico”, Prentice Hall.

[4] Martín de Brío, B. (2006). “Redes Neuronales y Sistemas Borrosos”, Ra-Ma.

[5] Matlab Help.

[6] Smith, S.T. (2006). “Matlab Advanced GUI Development”, Dog-Ear Publishing.